VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2016 Bc. Petr Pyszko

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Palubní jednotka automobilu v mobilním telefonu
Vehicle Monitor Unit in Mobil Phone

2016 Bc. Petr Pyszko

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Petr Pyszko**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Palubní jednotka automobilu v mobilním telefonu.
Vehicle Monitor Unit in Mobile Phone

Jazyk vypracování: čeština

Zásady pro vypracování:

Navrhněte palubní jednotku automobilu pro mobilní telefon, která se bude k řídícímu systému automobilu připojovat bezdrátově. Jednotka bude schopná pomocí protokolu OBD II monitorovat a zpracovávat údaje a dále je bude poskytovat mobilnímu zařízení, které zastoupí uživatelské rozhraní palubního počítače. Zohledněte také možnost diagnostiky řídicí jednotky.

1. Seznamte se s protokoly OBD II.
2. Porovnejte vlastnosti dostupných palubních jednotek.
3. Navrhněte vlastní zapojení jednotky pro komunikaci s OBD II a realizujte zapojení na nepájivém kontaktním poli.
4. Napište potřebné programové vybavení pro komunikaci a zpracování mezi řídicí jednotkou a mobilním zařízením.
5. Navrhněte vhodné uživatelské rozhraní pro mobilní zařízení a následně jej zrealizujte.
6. Otestujte jednotku s mobilním zařízení v provozu, ověřte měřené hodnoty porovnáním s originálním palubním počítačem.
7. Ověřte spolehlivost, stabilitu a popište vlastnosti, přínosy a možnosti diagnostiky.

Seznam doporučené odborné literatury:

[1] Reto Meier, Professional Android 4 Application Development, Wrox; 3 edition (May 1, 2012),ISBN: 1118102274
[2] http://www.obdii.com/
[3] http://developer.android.com/index.html

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Olivka, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 29.04.2016

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 26.4.2016

podpis studenta

Velké poděkování náleží především vedoucímu mé diplomové práce Ing. Petru Olivkovi, Ph.D. za ochotu, trpělivost, náměty, pohotové odpovědi na mnou kladené otázky a nakonec za veškerý čas strávený nad mou prací.

Abstrakt

Diplomová práce se zabývá implementací aplikace pro mobilní zařízení se systémem Android, která bezdrátově komunikuje přes rozhraní ELM327, jenž je propojeno s řídící jednotkou přes zásuvku OBD. Aplikace je náhradou palubních počítačů motorových vozidel s rozšířenými funkcemi. Práce se měla původně zabývat i návrhem zařízení, které by bylo trvale připojeno do zásuvky OBD, nicméně z důvodu nedostatku informací o časování se použilo rozhraní ELM327.

Klíčová slova

ELM327, OBD, OBDII, EOBD, Bluetooth, palubní počítač, PID, Android, ORM lite, řídicí jednotka, ECU, readiness

Abstract

This thesis deals with implementation of applications for mobile devices running Android, which communicates wirelessly via ELM327, which is connected to the control unit via the OBD socket. The application is a replacement board computer for motor vehicles with advanced features. Work was initially intended to deal with the design of equipment that would be permanently attached to the OBD socket however, because of a lack of information about the timing interface ELM327 was used.

Keywords

ELM327, OBD, OBDII, EOBD, Bluetooth, onboard computer, PID, Android, ORM lite, control unit, ECU, readiness

Obsah

Seznam symbolů a zkratek

Bluetooth – bezdrátová technologie pro komunikaci

ECU – Engine computer unit - řídicí jednotka

OBD – OnBoard Diagnostic - Palubní diagnostika

OS – Operation systém - Operační systém

PID – Protokol id - Id parametru

Readiness – kódy, které se využívají na emisích

UI – User interface – Uživatelské rozhraní

Wifi – bezdrátová technologie pro komunikaci na větší vzdálenosti

Seznam obrázků

Seznam tabulek

1 Úvod a cíl práce

V posledních patnácti letech jsou téměř všechna motorová vozidla vybavena protokolem OBD, který umožňuje komunikaci s řídicí jednotkou, avšak některá vozidla nemají potřebný palubní počítač, který by umožnil vyčítání informací o provozu vozidla. Pokud vozidlo disponuje palubním počítačem, pak je převážně realizováno pomoci malého displeje a poskytuje pouze základní informace, i když možnosti řídící jednotky jsou mnohem větší. Cílem práce je vytvořit zařízení s programem, které bude připojeno do zásuvky OBD a pomoci které bude komunikovat s řídicí jednotkou vozidla. Načtené údaje se budou zpracovávat, ukládat a dále budou poskytovány mobilnímu zařízení, které bude vybaveno aplikací v přehledném grafickém prostředí. Hlavním cílem je dlouhodobé ukládání dat a zobrazování aktuálních i statistických údajů, včetně možnosti diagnostiky.

2 Seznámení s protokolem OBD

OBD neboli palubní diagnostika pracuje na principu neustálého sledování čidel (ideálně ze všech systémů) a následném vyhodnocování naměřených dat. Pokud čidlo z některého systému vykazuje nesrovnalosti delší dobu, pak systém tuto komponentu označí jako chybnou.

Diagnostika je zabudována v řídící jednotce vozidla, která komunikuje s dalšími komponentami vozidla, ze kterých průběžně sbírá data.

V USA jsou protokolem OBDII povinně vybavena všechna vozidla, vyrobená od roku 1996. V Evropě je tento protokol označován jako EOBD („Europe On-Board Diagnostics“), ale jedná se o totožný protokol, který je povinný pro všechny nové modely s benzínovým motorem od roku 2001, respektive od roku 2003 pro všechny naftové motory [1].

Emisní systémy, které jsou neustále měřeny a vyhodnocovány:

- Katalyzátor.
- Vyhřívání katalyzátoru.
- Odvětrávaní palivové nádrže.
- Systém sekundárního vzduchu.
- Klimatizace.
- Lambda sondy.
- Ohřev lambda sondy.
- Recirkulace výfukových plynů EGR.

Tyto údaje, případně další, si pak můžeme skrze OBD zásuvku z řídící jednotky vyčíst a dále zpracovávat dle potřeby.

2.1 Definované normy protokolu OBD II

Protokol je definován několika normami:

- SAE J1962.
- ISO9141.
- SAE J1850.
- SAE J1979.
- ISO-15765.

2.1.1 SAEJ1962

Standard SAE J1962 stanovuje podobu 16-pinové zásuvky, která je umístěna v blízkosti řidiče pod volantem, pomocí níž komunikujeme s řídící jednotkou.

2.1.2 ISO 9141, SAE J1850

Normy definují povinnou fyzickou vrstvu pro komunikaci řídicí jednotky s emisními systémy.

2.1.3 SAE J1979, ISO-15765

Norma SAE J1979 definuje PIDy neboli kódy, pomoci kterých komunikujeme s řídicí jednotkou. Kódy jsou hodnoty v šestnáctkové sestavě a jsou složené ze dvou, nebo čtyř hexadecimálních znaků. První dva znaky označují režim (01-0A), další znaky se pak dělí dle skupiny.

Celkově existuje 10 skupin, režimů:

- 0x01 – Aktuální data.
- 0x02 – Zmražená data.
- 0x03 – Uložené chyby z diagnostiky.
- 0x04 – Vymazání chyb.
- 0x05 – Výsledky testů z lambda sondy.
- 0x06 – Výsledky testů z ostatních systémů vozidla.
- 0x07 – Výsledky testů z hlavních systémů vozidla.
- 0x08 – Kontrolní operace systémů.
- 0x09 – Informace vozidla (např. VIN).
- 0x0A – Dlouhodobé chyby.

První skupina (0x01) je nejrozsáhlejší. Obsahuje více než 100 kódů. Každý kód je definován dvojicí hexadecimálních hodnot, počtem návratových bajtů a transformací neboli rovnicí, pomocí které získáme z návratových bajtů čitelnou hodnotu ve správném tvaru. U většiny kódů se dá dohledat popis, minimální, maximální hodnota a jednotka. Kromě základních PIDů, které jsou vidět v tabulce 1, existují i další, které jsou definovány přímo výrobcem.

Tabulka 1: Ukázka důležitých PIDů z prvního a druhého režimu [2].

PID	Popis	Výpočet	Jednotka
0x0100	Podpora následující 32 PIDů	--	
0x0104	Vypočtené zatížení motoru	A*100/255	%
0x0105	Teplota chladicí kapaliny motoru	A-40	°C
0x010C	Otáčky motoru	(A*256+B)/4	Otáčky/min
0x010D	Rychlost vozidla	A	Km/h
0x010F	Teplota nasávaného vzduchu	A-40	°C
0x0110	Váha vzduchu	((A*25)+B)/100	g/s
0x0111	Poloha škrticí klapky	(A*100)/255	%
0x012F	Palivo v nádrži	(A*100)/255	%
0x013C	Teplota před katalyzátorem senzor č. 1	((A*256)+B)/10 – 40	°C
0x013D	Teplota za katalyzátorem senzor č. 1	((A*256)+B)/10 – 40	°C
0x0142	Napětí v síti	((A*256)+B)/1000	V
0x0144	Poměr paliva a vzduchu	((A*256)+B)/32768	poměr
0x0146	Teplota venkovního vzduchu	A-40	°C
0x015B	Nabití baterie u hybridních vozidel	(A*100)/255	%
0x015C	Teplota oleje v motoru	A-40	°C
0x015D	Časování vstřikování paliva	((A*256)+B)/128-210	°
0x015E	Spotřeba paliva	((A*256)+B)/20)	l/h
0x0163	Referenční krouticí moment	(A*256)+B	Nm
0x0141	Readiness kódy	Viz tabulka 2	

Poslední PID (0x0141) poskytuje informace o výsledcích testů, jedná se takzvané readiness kódy, jejich popis najdete v kapitole 5.4.10. Příkaz vrací 4 bajty, z těchto bajtů dle předpisu z tabulky

2 můžeme vyčíst, které testy jsou řídicí jednotkou podporovány a které jsou splněny. Bajty C a D znamenají třetí a čtvrtý bajt načtené z daného PIDu.

Tabulka 2: Readiness kódy [3] [4].

Název testu	Podporován	Nedokončen
Katalyzátor	C0	D0
Vyhřívání katalyzátoru	C1	D1
Odvětrávání palivové nádrže	C2	D2
Systém sekundárního přívodu vzduchu	C3	D3
Klimatizace	C4	D4
Lambda sonda	C5	D5
Vyhřívání lambda sondy	C6	D6
Zpětné vedení výfukových plynů (EGR)	C7	D7

Druhý režim (0x02)používá stejné kódy, jako předešlý první režim, nicméně vrací informace z doby, kdy se vyskytla poslední chyba vozidla (pokud je nějaká chyba uložena).

Třetí režim (0x03) vyčítá chyby, které jsou aktuálně uloženy v řídicí jednotce. Dle normy ISO-15765 se pro každou chybu načítá 6 bajtů, první dva bajty označují kód chyby ve tvaru X0123 a to dle následujících pravidel.

První znak je dán posledními dvěma bity prvního bajtu (A7-A6), kde význam znaku je popsán v tabulce 3.

Tabulka 3: Rozdělení chyb dle prvního znaku [2].

Bity	Zkratka	Část vozidla
00	P	Hnací ústrojí
01	C	Podvozek
10	B	Karosérie
11	U	Síť

Druhý znak je dán pátým a čtvrtým bitem (A5-A4), výsledné číslo je v rozmezí 0-3, hodnota 0 znamená všeobecné chyby a hodnoty 1-3 jsou určené pro chyby, dané výrobcem vozidla.

Třetí znak je dán třetí až nultým bitem (A3-A0), výsledné číslo je v rozmezí 0-F, nabývané hodnoty jsou však v rozmezí 1-C a s tímto významem:

1 - Dávkováni paliva a vzduchu.

2 - Dávkování paliva a vzduchu ve vstřikovacím cyklu.

3 - Zapalování .

4 - Emisní systém.

5 - Hlídání rychlosti vozidla a systém nečinnosti vozidla.

6 - Vnější obvody řídicí jednotky.

7, 8, 9 – Převodovka.

A, B, C – Hybridní pohon.

Další znaky jsou dány z druhého bajtu, a to bity B7-B4 pro první a bity B3-B0 pro druhý bajt.

3 Přehled dostupných jednotek a aplikací

Diagnostické jednotky se dělí do dvou základních skupin – diagnostické zařízení a diagnostické rozhraní.

Dražší diagnostické zařízení kromě kabelu, disponují převážně velkým displejem se zabudovaným softwarem poskytující plnohodnotné uživatelské rozhraní, včetně všech potřebných funkcí.

Druhá skupina, tedy rozhraní, jsou pouze převodníky, které zajišťují časování a předávaní informace mezi řídící jednotkou a koncovým zařízením (stolní počítač, notebook, mobilní zařízení).

Diagnostické zařízení se dále dělí do dvou základních skupin:

- Originální komerční.
- Neoriginální komerční.

3.1 Originální komerční diagnostické zařízení

Každá automobilka, případně koncern, předepisují autorizovaným servisům, jaké diagnostiky musí používat. Jedná se běžně o diagnostiky s cenou v řádu desetitisíců až statisíců.

Například automobilka Škoda Auto, případně koncern Volkswagen předepisují diagnostiku VAS 5052, která je zobrazena na obrázku 1. (případně VAS 5051B (obrázek 2) s cenou okolo 60 tisíc Kč nebo starší VAG 1552), automobilka BMW předepisuje Modic a Mercedes StarDiagnose.

Mezi hlavní **nevýhody** patří nemožnost aktualizace (alespoň u starších typů), tedy novější vozidla starší diagnostikou nemusí být podporovány.

Naopak hlavní **výhodou** je bezesporu možnost vyčíst veškeré možné údaje, včetně nejrůznějších možností konfigurací, které neoriginální diagnostiky podporovat nemusí. Další výhodou jsou předem připravené manuály a záruka spolehlivosti.

Obrázek 1: Diagnostické zařízení VAS 5052 [5].

Obrázek 2: Diagnostické zařízení VAS 5051B [6].

3.2 Neoriginální diagnostické zařízení

Další alternativou je využití jiných dostupných neoriginálních diagnostik, které jsou levnější, ale poskytují často stejné funkce. Převážně je využívají neautorizované servisy, které se nesoustředí pouze na jednu značku, ale na velké spektrum značek.

3.3 Diagnostické rozhraní se softwarem

Další přívětivější alternativou je využít pouze rozhraní, např. kabel s převodníkem, kdy uživatelské rozhraní zajišťuje nejčastěji notebook nebo stolní počítač.

Rozhraní se dělí na univerzální a na ty, které se soustředí na určitou značku, případně skupinu značek.

Hlavní výhody:

- Cena - převážně stačí zakoupit kabel, software je běžně ke stažení nebo je dodán na přenosném médiu.
- Uživatelské rozhraní – pracujeme na námi známem OS, snadná přenositelnost exportovaných dat.
- Aktualizace softwaru na novější verze.

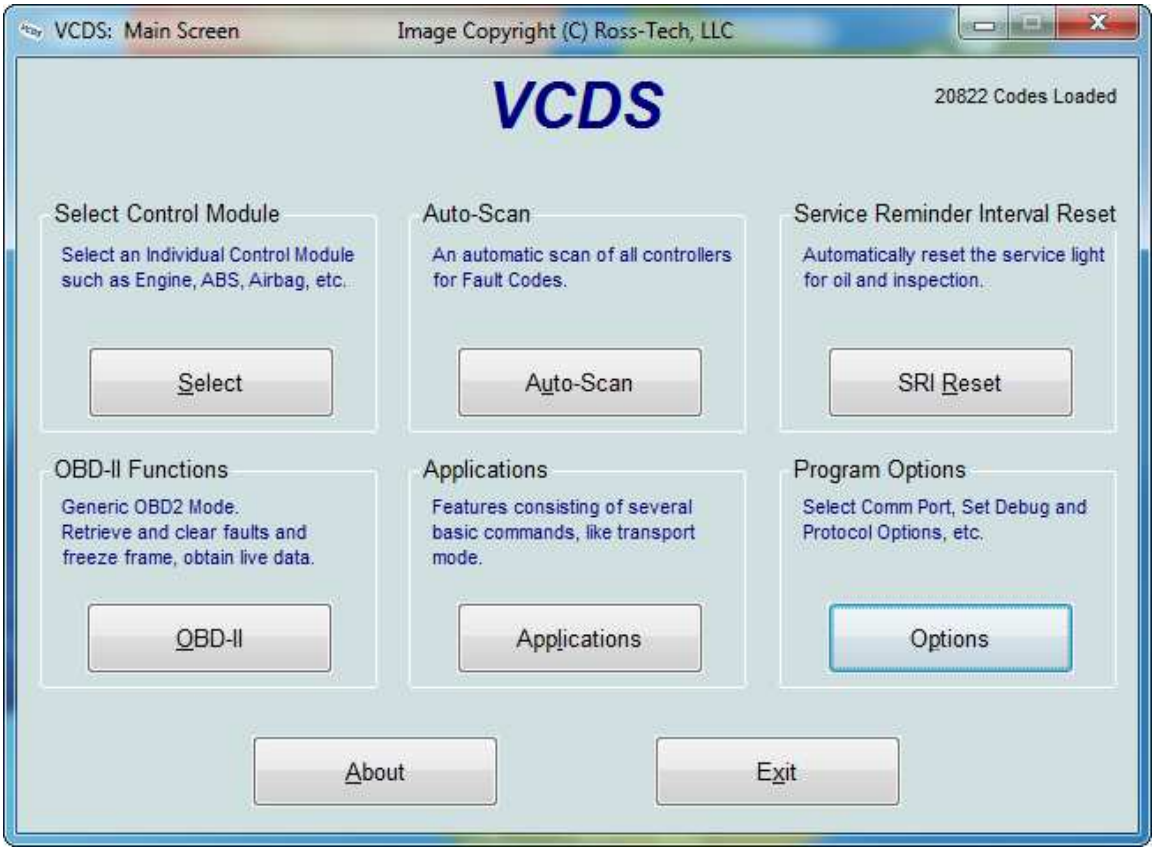
3.4 Sériová diagnostika VAG-COM

V tuzemsku je asi nejrozšířenější alternativou diagnostický kabel s využitím softwaru VAG-COM, který podporuje všechny vozidla koncernu VW Group (Škoda, Volkswagen, Audi, Seat).

Obrázek 3: OBD kabel s převodníkem z RS232 na USB.

První verze komunikovaly přes zastaralý port COM. V dnešní době se kabely vyrábějí standardně s koncovkou USB, jež je znázorněn v obrázku 3.

Největší výhodou softwaru je neustály vývoj a poskytování nových verzí, které vycházejí prakticky každý rok.

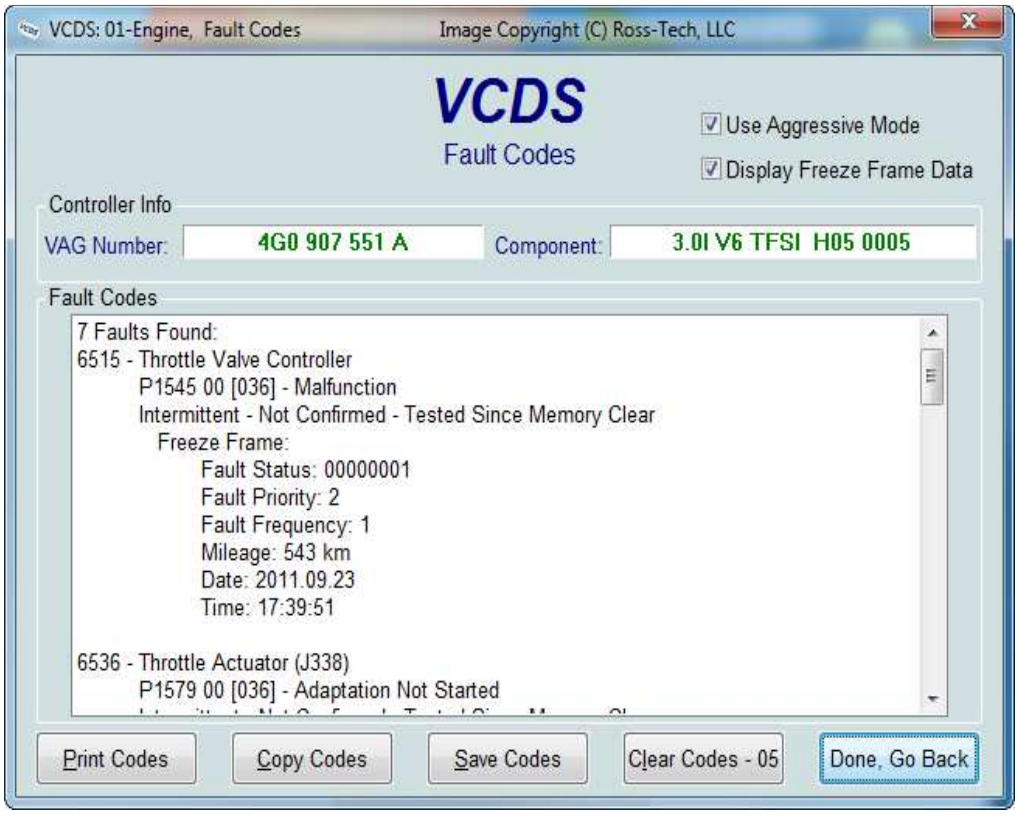


Obrázek 4: Rozcestník programu VAG-COM [7].

Na obrázku 5 lze vidět jednu z nejdůležitějších funkcí, kterou je výčet uložené chyby.

Informace které jsou o chybě poskytnuty:

- Kód chyby.
- Název chyby.
- Popis chyby.
- Vážnost a četnost chyby.
- Datum a čas chyby.
- Ujetá vzdálenost od uložení chyby.



Obrázek 5: Výčet chyby v programu VAG-COM [8].

3.5 Univerzální sériová diagnostika ELM 327

Tato diagnostika, zobrazená na obrázku 6 je ve světě nejrozšířenější rozhraní, které umí komunikovat až s 10 protokoly a umožňuje několik vlastních definic časování. Existuje ve více alternativách s možností připojení přes USB kabel, bezdrátový bluetooth nebo wifi.

Rozhraní poskytuje množství příkazů pro nastavení komunikace, jako je maximální prodleva, druh protokolu, automatické hledání protokolu, hlavička, apod.

Dle nastavení pak přeposílá příkazy přes vybraný protokol do řídicí jednotky a jejich výsledky poskytuje zpátky koncovému zařízení. Jelikož se jedná o nejrozšířenější diagnostické rozhraní s velkou podporou komunity, existuje spousta aplikací a to nejen pro operační systém Windows nebo Linux, ale také pro operační systémy mobilních zařízení.

Hlavní přednostní diagnostiky je cena, která se pohybuje od 100 Kč a výše [9].

Obrázek 6: Rozhraní ELM327 [9].

Pro operační systém Android existuji desítky dostupných aplikací, mezi nejznámější patří Torque Pro a OBD Car Doctor, nicméně jsem nenalezl žádnou aplikaci, která by plně vynahradila palubní počítač.

3.6 Android aplikace Torque Pro

Torque Pro na obrázku 7 představuje nejrozšířenější aplikaci pro systém Android, její cena se pohybuje okolo 90 Kč. Vývojáři poskytují i verzi Torque Lite, která je zdarma, nicméně má některé funkce omezené, avšak pro základní diagnostiku a výčet měřených veličin je dostačující. Funkce aplikace jsou následující:

- Zobrazení aktuálně měřených veličin.
- Možnost měření aktuálních veličin ve formě grafu.
- Zobrazení uložených chyb a jejich vymazání.
- Zobrazení posledních jízd vozidla – vzdálenost, čas, spotřeba a trasa.
- Výsledky testů z readiness kodů.

Největším nedostatkem aplikace je absence dlouhodobého měření veličin a možnost přidávat měření s možnosti nulování.

Obrázek 7: Aplikace Torque Pro [10].

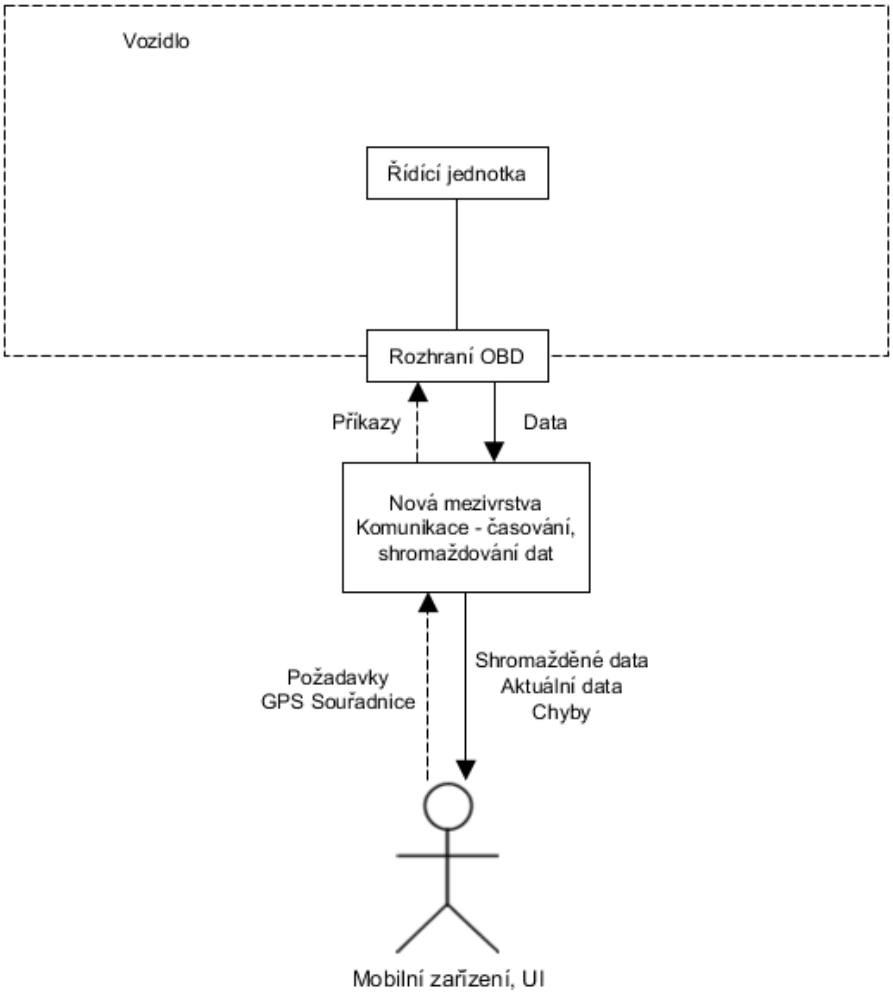
4 Návrh vlastní diagnostické jednotky

Cílem práce je navrhnout zařízení, umožňující komunikaci mezi řídící jednotkou a koncovým zařízením, v tomto případě mezi mobilním telefonem s operačním systémem Android 4.0 a výše.

4.1 Analýza požadavků pro mezivrstvu

Požadované vlastnosti:

- Podpora protokolu ISO 15765-4 CAN (11 bit ID, 500 kbaud) případně dalších protokolů.
- Ukládaní naměřených údajů a následné přiřazování k měřičům a jízdám.
- Možnost diagnostiky – vyčítání uložených chyb a jejich mazání.
- Poskytování dat mobilnímu zařízení pomocí bezdrátového spojení.
- Ukládání GPS souřadnic v případě spojení s mobilním zařízením.
- Definice vlastních veličin, které budou měřeny.
- Minimalizované rozměry – krabička s výstupem patice, která svými rozměry nebude bránit v zasunutí do OBD patice.
- Velký úložný prostor.



Obrázek 8: Umístění nového zařízení.

4.2 Hardwarový návrh jednotky

Z výše zmíněných požadavků je patrné, že součásti jednotky bude patice pro paměťovou kartu, patice pro OBD zásuvku, bluetooth modul a mikročip, který bude obstarávat funkčnost. Jelikož mám zkušenosti s mikročipem Atmega od společnosti Atmel, rozhodl jsem se na tomto mikročipu jednotku postavit. Pro bezdrátovou komunikaci bude použitá komponenta HC-05 Bluetooth. Pro spojení s paměťovou kartou a OBD zásuvkou již žádné moduly nepotřebujeme, protože patice se pájí přímo na konektory mikročipu.

Tabulka 4: Výčet cen základních součástek pro navrhované zařízení.

Název	Cena v ČR (ze zdroje http://www.heureka.cz)	Cena v zahraničí (ze zdroje http://www.ebay.com)
Atmega 128kb	149 Kč	25 Kč
HC-05 Bluetooth	245 Kč	90 Kč
Paměťová karta 2 GB	100 Kč	70 Kč

Výčet základních součástek můžete vidět v tabulce 4. Součet všech součástek v zařízení včetně patic, desky a obalu by se pak pohybovaly do 500 Kč. Nicméně z důvodu cenově dostupnějších zařízení a hlavně z důvodu nedostatku informací o časování s řídicí jednotkou bylo nutné použít dostupné rozhraní, které řeší časování. Logika zpracování dat se tak přesune do mobilního zařízení s operačním systémem Android, mobilní aplikace se tak rozdělí na dvě části.

1. Aplikace, která bude řešit komunikaci a zpracování dat a jejích ukládání.
2. Uživatelské rozhraní pro zobrazování dat s možnosti nastavení.

Po analýze dostupných jednotek jsem zvolil rozhraní ELM 327.

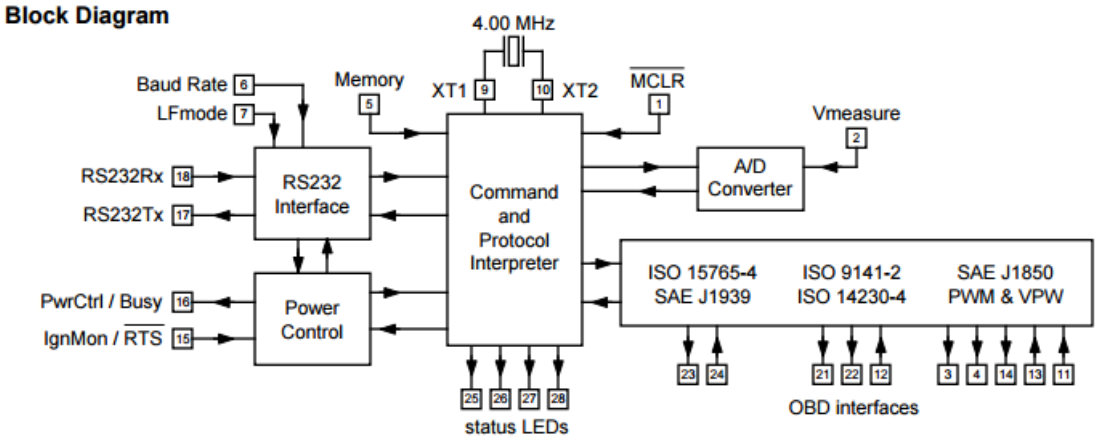
Hlavní důvody výběru rozhraní ELM327

- Jednotka je volně dostupná na portálech http://ebay.com nebo http://aliexpress.com v různých variantách.
- Většina jednotek podporuje bezdrátovou komunikaci přes wifi nebo bluetooth.
- Jsou dostupné v ceně již od 100 Kč včetně dopravy.
- Obsah jednotky je zabudovaný přímo v patici, takže ve vozidle nemusíme přetahovat kabely.
- Možnost použití jiných dostupných aplikací.

Hlavní výhodou pro uživatele navrhované aplikace je nezávislost použitého zařízení, takže nebudou nijak vázaní na použitý hardware, ale můžou používat běžně dostupné jednotky.

4.3 Popis komunikace ELM 327

Z blokového schéma na obrázku 9 je vidět, že mikročip pracuje na frekvenci 4 MHz a komunikuje s 5ti standardy. Z toho dva (ISO 15768-4, SAE J1939) komunikují po dvou drátech, další dva (ISO 9141-2, ISO 14230-4) po třech vodičích a poslední standard (SAE J180 PWM & VPW) komunikuje přes 5 vodičů.



Obrázek 9: Schéma jednotky ELM327.

Tyto standardy mají různé alternativy časování a šířku komunikace.

Všechny podporované protokoly jednotkou ELM327:

- SAE J1850 PWM (41.6 kbaud)
- SAE J1850 VPW (10.4 kbaud)
- ISO 9141-2 (5 baud init, 10.4 kbaud
- ISO 14230-4 KWP (5 baud init, 10.4 kbaud)
- ISO 14230-4 KWP (fast init, 10.4 kbaud)
- ISO 15765-4 CAN (11 bit ID, 500 kbaud)
- ISO 15765-4 CAN (29 bit ID, 500 kbaud)
- ISO 15765-4 CAN (11 bit ID, 250 kbaud)
- ISO 15765-4 CAN (29 bit ID, 250 kbaud)
- SAE J1939 CAN (29 bit ID, 250* kbaud)

Kromě zmiňovaných protokolů, rozhraní umožňuje komunikaci i pomocí uživatelem definovaného časování.

S ELM327 komunikujeme buď bezdrátově, nebo přes kabel. Při komunikaci se s rozhraním dorozumíváme pomocí příkazů, příkazy jsou dvojího typu.

První sada příkazů obsahuje ty, které používá přímo rozhraní ELM327. Příkazy začínají prefixem „AT“ a slouží k inicializaci spojení s řídicí jednotkou, případně dodefinování komunikace. Základních příkazů je okolo 50ti, pak existuje dalších cca 50 příkazů, které jsou určené pro jednotlivé protokoly. Ukázka některých příkazů je vidět v tabulce 5.

Tabulka 5: Ukázka příkazů ELM327 s počátečními písmeny AT [11].

Příkaz	Popis
AT SP 0	Nastavení a uložení automatického hledání protokolu.
AT SP Ah	Automatické hledání protokolu s prioritním protokolem „h“.
AT SP h	Nastavení protokolu h – 1 až A tedy 10 protokolů + další definovatelné B C.
AT TP h	Vyzkoušení protokolu „h“.
AT TP Ah	Vyzkoušení protokolu „h“ nebo automatické hledání
AT Z	Tovární nastavení
AT SH xy	Nastavení hlavičky na text „xy“ – pokud chceme komunikovat s jinými komponentami vozidla
AT MA	Zapnutí monitorování
AT SR hh	Ukládání odpovědí do paměti na adresu „hh“
AT SD bb	Zapsání bajtu „bb“
AT RD	Načtení uložených dat
AT <CR>	Opakování minulého příkazu
AT LP	Zapnutí úsporného režimu
AT H0	Vypnutí hlaviček
AT H1	Zapnutí hlaviček
AT E0	Vypnutí hlavičky odpovědí
AT E1	Zapnutí hlaviček odpovědí
AT S0	Vypnutí mezer
AT S1	Zapnutí mezer

Nejdůležitější příkaz je nastavení protokolu, případně jeho automatické vyhledání. Po úspěšném spojení můžeme začít komunikovat s řídící jednotkou pomocí OBD příkazů.

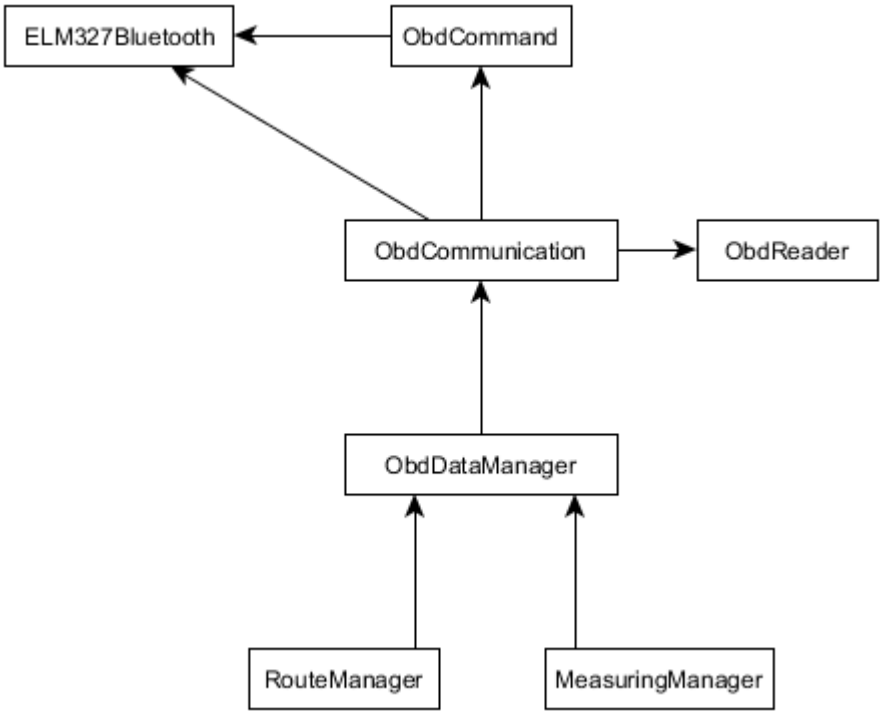
Druhá sada příkazů jsou OBD příkazy. Tyto příkazy se posílají ve formátu „XXYY“ kde první dva znaky znamenají režim, např. 01, druhá dvojice pak znamená konkrétní PID, např. 2F, tedy pokud ho daný režim vyžaduje. ELM327 podporuje i formát „XXYYZ“, kde „Z“ znamená počet očekávaných bajtů na výstupu. ELM327 podporuje i vícenásobné zasílání příkazů najednou, pokud takovou funkci daný protokol dovoluje. Bližší popis OBD příkazů najdete v kapitole 2.1.3.

4.4 Programové vybavení pro komunikaci

Cílem komunikace je dotazování řídící jednotky pomocí příkazů, která bude vracet potřebné údaje pro palubní počítač. Nejdůležitější je měření rychlosti a s tím spojená ujetá vzdálenost a spotřeba. Dále se budou měřit veškeré veličiny, které poskytuje řídící jednotka a uživatel si je nadefinuje pomocí UI.

Jak je vidět na obrázku 10, komunikace a zpracování dat je rozděleno do několika komponent, které jsou dále popsány.

Následující části jsou psány v jazyce Java pro operační systém Android, krátký popis najdete v kapitole 5.1.



Obrázek 10: Rozdělení zpracování dat v několika komponentách.

4.4.1 Komponenta ELM327Bluetooth

Komponenta řídí komunikaci mezi rozhraním ELM327 a aplikací. Nejdříve se připojí přes bluetooth a po úspěšném spojení zkouší spojení s řídicí jednotkou dle zvoleného protokolu, případně

se testují všechny dostupné protokoly, dokud řídicí jednotka neodpoví. Tuto funkci testování všech protokolů podporuje přímo i rozhraní ELM327.

```
new ObdCommand(this,"ATD").Execute();
new ObdCommand(this,"ATZ").Execute();
new ObdCommand(this,"AT E0").Execute();
new ObdCommand(this,"AT AT0").Execute();
new ObdCommand(this,"AT L0").Execute();
new ObdCommand(this,"AT S0").Execute();
new ObdCommand(this,"AT H0").Execute();
new ObdCommand(this,"AT SP 0").Execute();
```

Ukázka kódu 1: Ukázka inicializace rozhraní ELM327 z aplikace.

- ATD – Defaultní nastavení
- ATZ – Resetování zařízení
- AT E0 – Vypnutí hlavičky odpovědi
- AT AT0 – Vypnutí variabilního časování
- AT L0 – Vypnutí dodatečných řádků
- AT S0 – Odpovědi mezer
- AT H0 – Vypnutí hlaviček
- AT SP 0 – Nastavení automatického hledání protokolu a uložení

Jakmile je spojení s řídicí jednotkou úspěšně navázáno, tedy odpovídá na OBD příkazy, komponenta nastaví stav připojeno a poskytne komunikační kanál pro další komunikaci.

4.4.2 ObdCommand

Základní jednotkou komunikace je příkaz. Příkaz je kód, který je zaslán do řídící jednotky a ta dle kódu odešle zpět celočíselné hodnoty, které musí být transformovány dle předpisu, část předpisu můžete najít v tabulce 1.

Příkazy se zasílají přes vytvořený komunikační kanál komponenty ELM327Bluetooth.

```
protected void SendCommand() {
try {
IObdConnection.GetOutupStream().write(command.getBytes());
IObdConnection.GetOutupStream().flush();
} catch (Exception e) {
if ( IObdConnection.IsConnected() ) {
IObdConnection.Interupted();
}
isNumberResult = false;
executed = false;
}
}
```

Ukázka kódu 2: Komunikace s rozhraním ELM327, zaslání příkazu.

```
protected void ReadData()
{
byte c = 0;
this.buffer.clear();
this.integerBuffer.clear();
try {
while ((char)(c = (byte) IObdConnection.GetInputStream().read()) != '>') {
buffer.add(c);
}
    } catch (Exception e) {
if ( IObdConnection.IsConnected() ) {
IObdConnection.Interupted();
}
isNumberResult = false;
executed = false;
return;
}…
}
```

Ukázka kódu 3: Načtení hodnoty z rozhraní ELM327.

Nejčastěji jsou v odpovědi jedna až čtyři celočíselné hodnoty a to vždy v rozmezí 0 až 255.

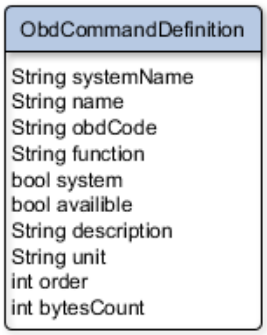
Pokud řídicí jednotka daný příkaz nepodporuje, pak neodpoví vůbec. Každý požadavek má definovanou prodlevu „čas vypršení“, tzn., jak dlouho čeká, než ECU odpoví. Pokud řídicí jednotka neodpoví, pak rozhraní ELM327 vrátí hodnotu „NODATA“. Čas vypršení může být definován předem, např. 250 milisekund, nebo může být variabilní.

Variabilní časování se přizpůsobuje dle odpovědí posledních požadavků, pokud ECU často neodpovídá ve stanovené lhůtě, pak se lhůta pro příští požadavek prodlouží. Pokud ale odpovídá v řádné době, pak se maximální lhůta čekání snižuje.

Řídicí jednotce trvá určitý čas, než požadavek zpracuje a odpoví. Čas je proměnlivý dle typu požadavku a délky odpovědi. Zpravidla trvá zpracování déle u požadavku s delším počtem bajtů v odpovědi. Také záleží na zdroji informací, odkud řídicí jednotka veličiny načítá. Pokud čteme hodnotu, která je uložena přímo v řídicí jednotce, potom je prodleva odpovědí krátká, jakmile čteme informace, které se v čase mění a řídicí jednotka se pro odpověď musí dotazovat do dalších komponent, např. do převodovky, poté je prodleva delší.

Čas odpovědí jednoho příkazu protokolu „ISO 15765-4 CAN (11 bit ID, 500 kbaud)“, který jsem testoval, se nejčastěji pohyboval v rozmezí 40 až 100 ms.

V aplikaci je předefinovaných necelých 100 příkazů, každý příkaz má předem definované chování, uživatel však má možnost existující příkazy upravit nebo přidat další.



Obrázek 11: Ukázka třídy ObdCommandDefnition.

Popis některých vlastností třídy z obrázku 11:

ObdCode – kód, který je složen ze dvou části, první část je kód skupiny (Viz 2.1 norma SAE), druhá část označuje číslo konkrétního PIDu (hlavně pro skupinu 01, tedy aktuálně měřené veličiny).

Function – každý kód má předem definovanou výpočetní funkci, pomocí které získáme správnou hodnotu veličiny. Výpočetní funkce se zadává v syntaxi javascriptu.

Ukázka funkce pro výpočet hodnoty otáček za minutu:

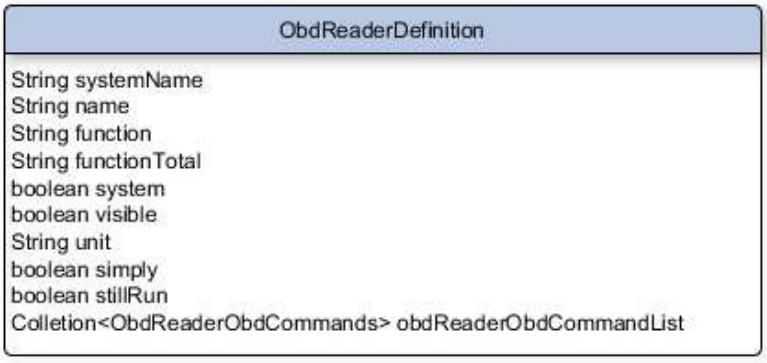
return ((a*256)+b)/4.0;

Availible – zde je uložena informace o tom, zdali aktuálně používané vozidlo podporuje daný příkaz.

BytesCount – počet bajtů, který očekáváme na výstupu.

4.4.3 ObdReader

Řídicí jednotka prostřednictvím příkazů sice vrací nejrůznější veličiny, ale ne vždy jsou to veličiny, které potřebujeme ke správné funkčnosti palubního počítače. Čítač umožňuje sloučit více příkazů do jedné výpočetní funkce, která vypočítá potřebnou hodnotu.



Obrázek 12: Ukázka třídy ObdReaderDefinition.

Popis některých vlastností třídy z obrázku 12:

Function – výpočetní funkce pro aktuální data např. litry za hodinu.

FunctionTotal – dlouhodobá výpočetní funkce např. spotřeba v litrech. Funkce je převážně totožná s funkcí pro výpočet aktuálních dat. Tato funkce se vždy násobí časovým rozdílem mezí poslední a předposledním spuštěním čítače v sekundách.

StillRun – příznak čítače, který běží neustále na pozadí.

ObdReaderObdCommandList – skupina příkazů, kde každý příkaz je rozšířen o parametr priority, který udává, jak často se má daný příkaz spouštět, tedy jeden příkaz se může spouštět častěji než druhý.

Názorná ukázka použití čítače je výpočet aktuálně spotřebovaného paliva, kdy pouze některá vozidla odpovídají na dotaz s informací o aktuální spotřebě, a proto musíme aktuální spotřebu dopočíst z jiných dostupných veličin. Jedna z možností je výpočet pomocí váhy vzduchu. Váha vzduchu je čidlo, které je umístěné v sacím potrubí a měří, kolik gramů vzduchu za sekundu proudí do motoru. Ideální poměr vzduchu a paliva je pak 14,7 u benzínu, případně 14,6 u nafty. Poté přepočteme váhu paliva na objem, kdy hustota benzínu je cca 0,75 g/cm^3, respektive 0,84 g/cm^3 hustota nafty. Poté hodnotu převedeme z mililitrů (cm^3) na litry, tedy vydělíme hodnotou 1000 a sekundy převedeme na hodiny vynásobením hodnotou 3600. Jelikož ne vždy je při výbuchu ve válci ideální poměr vzduchu a paliva, je vhodné hodnotu vynásobit bohatosti směsi, kterou získáme z lambda sondy. Řídicí jednotka poskytuje i „přesný“ vypočtený poměr, který můžeme použít.

Výsledná rovnice pro benzínové motory pak vypadá takto:

$$\begin{aligned}\textbf{Aktuální spotřeba} &= \textbf{Váha vzduchu} \\ &\times \textbf{Poměr vzduchu k palivu} \,/\, \mathbf{14,7} \,/\, \mathbf{0,75} \\ &\times \mathbf{3.6\ (l/h)}\end{aligned} \tag{4.1}$$

Pokud chceme převést hodnotu z l/h na l/100km, pak stačí výslednou hodnotu vynásobit rychlostí a vydělit hodnotou 100, viz rovnice 4.2.

$$\textbf{Aktuální spotřeba k rychlosti vozidla = Aktuální spotřeba / Rychlost * 100 (l/100km)} \tag{4.2}$$

Pokud dané vozidlo poměr vzduchu a paliva neposkytuje, pak se proměnná nahradí hodnotou 1.

Pokud vozidlo nedisponuje váhou vzduchu nebo její hodnotu neposkytuje, pak se hodnota váhy vzduchu dá vypočíst i z **dalších dostupných informací a rovnice:**

- Teplota nasávaného vzduchu ve stupních Kelvina.

- Absolutní tlak v sacím.
- Otáčky motoru.
- Účinnost motoru (převážně okolo 85 %).
- Zdvihový objem (převážně v rozmezí 1-3 litrů).
- Molární hmotnost vzduchu 28,97 g/mol.
- Molární plynová konstanta 8.314 J/K/mol.

$$\textbf{Váha vzduchu} = (\textbf{otáčky motoru} \times \textbf{tlak v potrubí} / \textbf{teplota saní} / \mathbf{120}) \times \textbf{účinnost motoru} \times \textbf{zdvihový objem} \times \textbf{molární hmotnost vzduchu} / \textbf{molární plynová konstanta}\ [12] \tag{4.3}$$

Další možnost výpočtu spotřeby je použití hodnoty z vypočteného aktuálního vytížení motoru. Tuto hodnotu vynásobíme maximální možnou spotřebou, respektive maximálním možným průtokem paliva a získáme aktuální spotřebu. V praxi se mi tato metoda osvědčila oproti první metodě lépe.

Výčet PIDů použitých v předešlých metodách:

- Vypočtené zatížení motoru – 0x0104.
- Absolutní tlak v sacím ve stupních – 0x010B.
- Otáčky motoru – 0x010C.
- Rychlost vozidla – 0x010D.
- Teplota nasávaného vzduchu – 0x010F.
- Váha vzduchu – 0x0110.
- Poměr paliva a vzduchu – 0x0144.

4.4.4 ObdCommunication

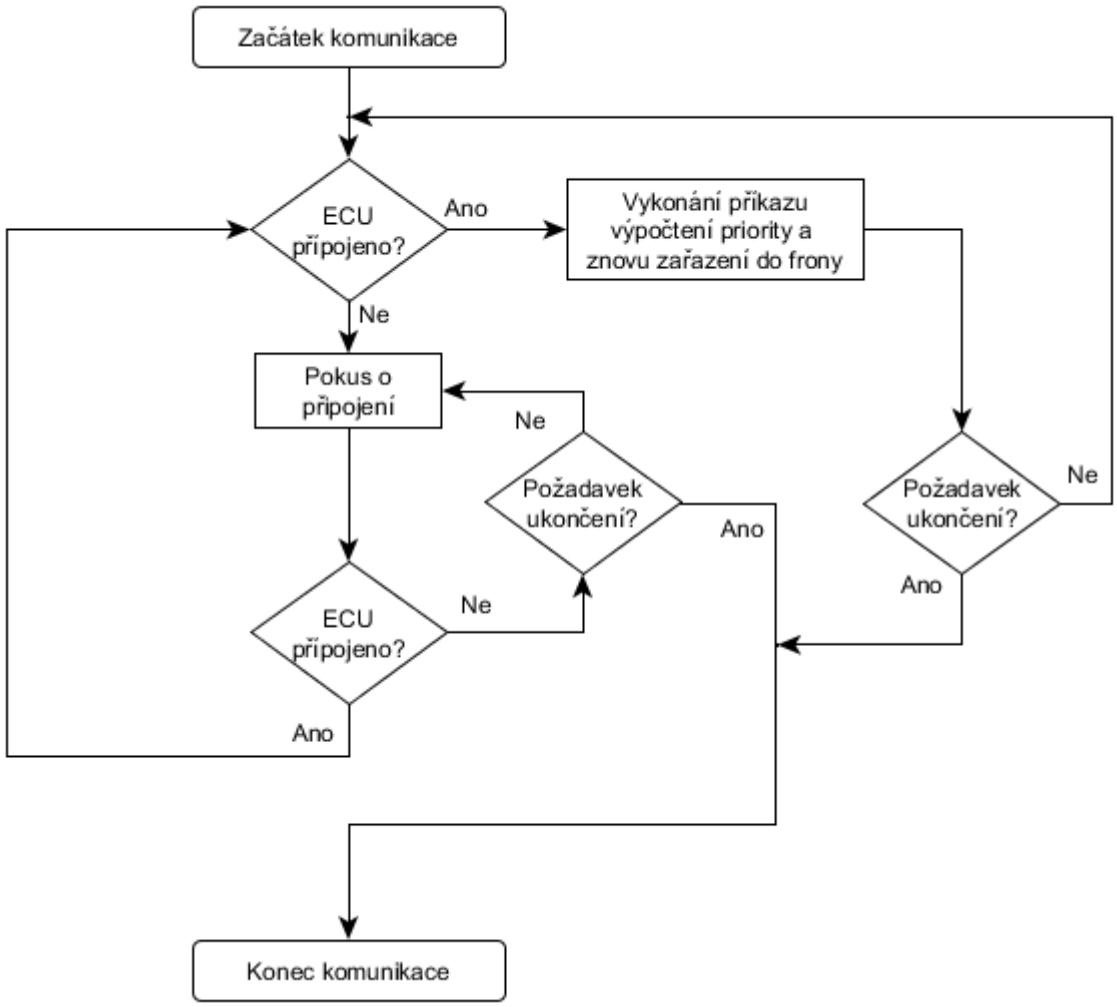
Tato komponenta se stará o správu čítačů a příkazů, které mají být aktuálně zasílány do řídicí jednotky, včetně volání inicializace spojení z komponenty ELM327Bluetooth.

Po inicializaci spojení a korektním spojením s řídící jednotkou, dochází k zasílání příkazů z jednotlivých čítačů. V jednom cyklu se vždy provede zavolání jednoho příkazu, poté obeznámení čítačů o změně stavu, přepočtení priority a znovu zařazení do fronty dle priority.

O komunikaci se stará komponenta ELM327Bluetooth, která se volá buď napřímo při inicializaci, nebo přes příkazy, které sdílejí vytvořený komunikační kanál s touto komponentou.

Komponenta se stará o sdílení výsledku z příkazů mezi čítači, tedy pokud stejný příkaz používá více čítačů, pak se příkaz volá pouze jednou a výsledek poskytne všem čítačům.

Priorita neboli prodleva do nejbližšího dalšího spuštění, se počítá dle nejnižší hodnoty ze všech čítačů nastavená k danému příkazu, které používají stejný příkaz. K vypočtené hodnotě se přidává skutečný čas odpovědi z posledního volání. Tím se ve frontě zvýhodní příkazy s malou prodlevou.



Obrázek 13: Životní cyklus komunikace mezi řídicí jednotkou a rozhraním ELM327.

V případě chyby, při vykonání dotazu dojde k uložení chybového stavu a tím se spustí inicializace spojení znovu. Po každém provedeném cyklu se testuje požadavek o ukončení komunikace.

Do fronty příkazů se mohou v průběhu komunikace registrovat další čítače nebo naopak odebírat. Pokud dojde k odebrání čítače z komunikace, pak jeho příkazy ve frontě sice zůstanou, ale po jejich dalším vykonání dochází k výpočtu priority příkazů, kde se buď zjistí, že už žádný čítač tento příkaz nepotřebuje a dojde k jeho vyloučení, nebo příkaz zůstane v rámci jiného čítače ale s upravenou prioritou.

4.4.5 ObdDataManager

Komponenta řeší správu všech příkazů a čítačů včetně ukládání jejich definic do databáze, případně umožňuje přidávat další definice čítačů a příkazů.

Dále shromažduje data o poslední a aktuální jízdě a o všech měřičích s příznakem „stillRun“ a dále je poskytuje uživatelskému rozhraní. Poskytované hodnoty můžou být buď minimální, maximální, celkové, průměrné nebo aktuální.

4.4.6 RouteManager

Manažer jízd ukládá data z jednotlivých jízd vozidla. Dále tyto informace poskytuje uživatelskému rozhraní. Jeho hlavní činností je měření všech základních údajů pro každou jízdu zvlášť, včetně měření dat ze všech čítačů s příznakem „stillRun“. Neustále sleduje základní veličiny, a jakmile je vozidlo nastartováno spustí měření nové jízdy. Pokud je pak motor po dobu „X“ sekund vypnutý, vyhodnotí jízdu za ukončenou a zapíše ji do databáze. Hodnota „X“ se načítá z nastavení.

Základní informace, které jsou vždy měřeny a ukládány:

- GPS souřadnice započetí a ukončení jízdy.
- Čas.
- Spotřeba, aktuální maximální spotřeba.
- Ujetá vzdálenost, maximální rychlost.
- Datum a čas započetí a ukončení jízdy.

Z těchto informací se pak dají vypočítat hodnoty:

- Průměrná rychlost.
- Průměrná spotřeba na 100 km, případné za hodinu.

Kromě těchto údajů, se měří a ukládají hodnoty ze všech čítačů, včetně jejich minimálních, maximálních a průměrných údajů.

Dále komponenta měří dle nastavených intervalů jednotlivé úseky jízdy. Intervaly jsou standardně nastaveny po 100 m a po 10 sekundách, případně to, co nastane dříve. Tyto hodnoty můžou být upraveny v nastavení.

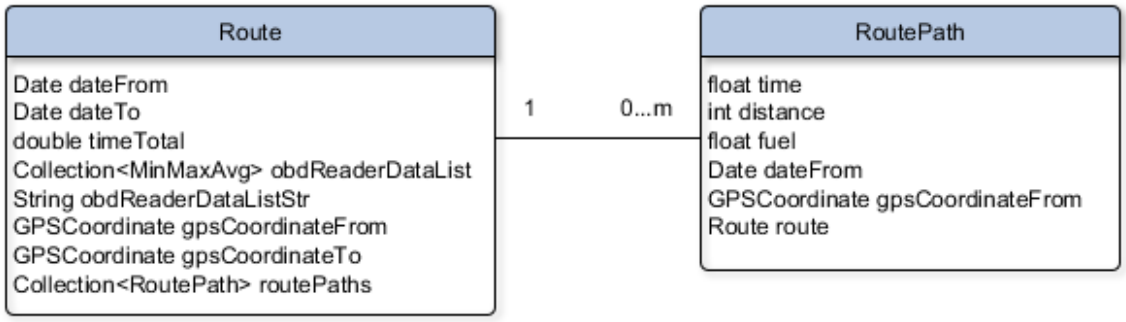
Pro tyto úseky se měří:

- GPS souřadnice začátku úseku.
- Čas.
- Spotřeba.
- Vzdálenost.
- Datum a čas kdy úsek nastal.

Z těchto informací se pak dá vypočítat:

- Průměrná rychlost v úseku.
- Průměrná spotřeba z úseku na 100 km nebo za hodinu.

Při ukládaní jednotlivých úseků se neukládají informace z ostatních čítačů nebo maximální rychlost apod., protože těchto záznamů bude v databázi uloženo i několik miliónů. Při ujetí vzdálenosti 100 kilometrů, by se pak při standardním nastavení intervalů uložilo 1000 záznamů. Jeden záznam má velikost necelých 100 bajtů, takže při ujetí takové vzdálenosti vznikne záznam o velikosti cca 100 kilobajtů. Při ujetí 100 000 kilometrů je to pak 100 megabajtů.



Obrázek 14: Ukázka tříd Route a RoutePath.

4.4.7 MeasuringManager

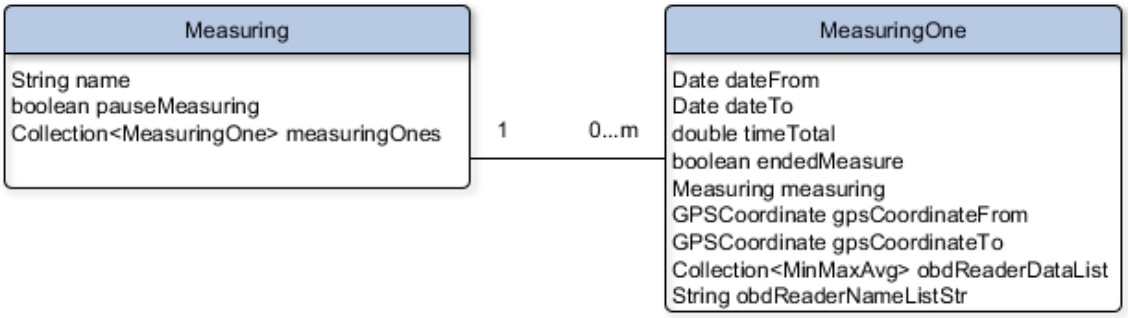
Manažer měřičů měří údaje pro jednotlivé měřící cykly. Pod pojmem měřič si můžeme představit měřidlo kilometrů ve vozidle. Tyto měřiče mají výhodu v tom, že kromě ujeté vzdálenosti měří spoustu dalších údajů. Při každém resetu, vynulováni, se dosažené hodnoty uloží do databáze a poté se měřiče vynulují.

Pro každý měřič se měří a ukládá do historie:

- Datum započetí a vynulování měřiče.
- Celkový čas.
- GPS souřadnice z doby započetí a vynulování měřiče.

K těmto údajům se přidávají veškeré údaje z čítačů s příznakem „stillRun“.

Manažer umožňuje nadefinovat neomezený počet měřičů a historie při resetování není nijak omezena. Každý měřič dále nese svůj název a dá se také zastavit nebo znovu povolit.



Obrázek 15: Ukázka tříd Measuring a MeasuringOne.

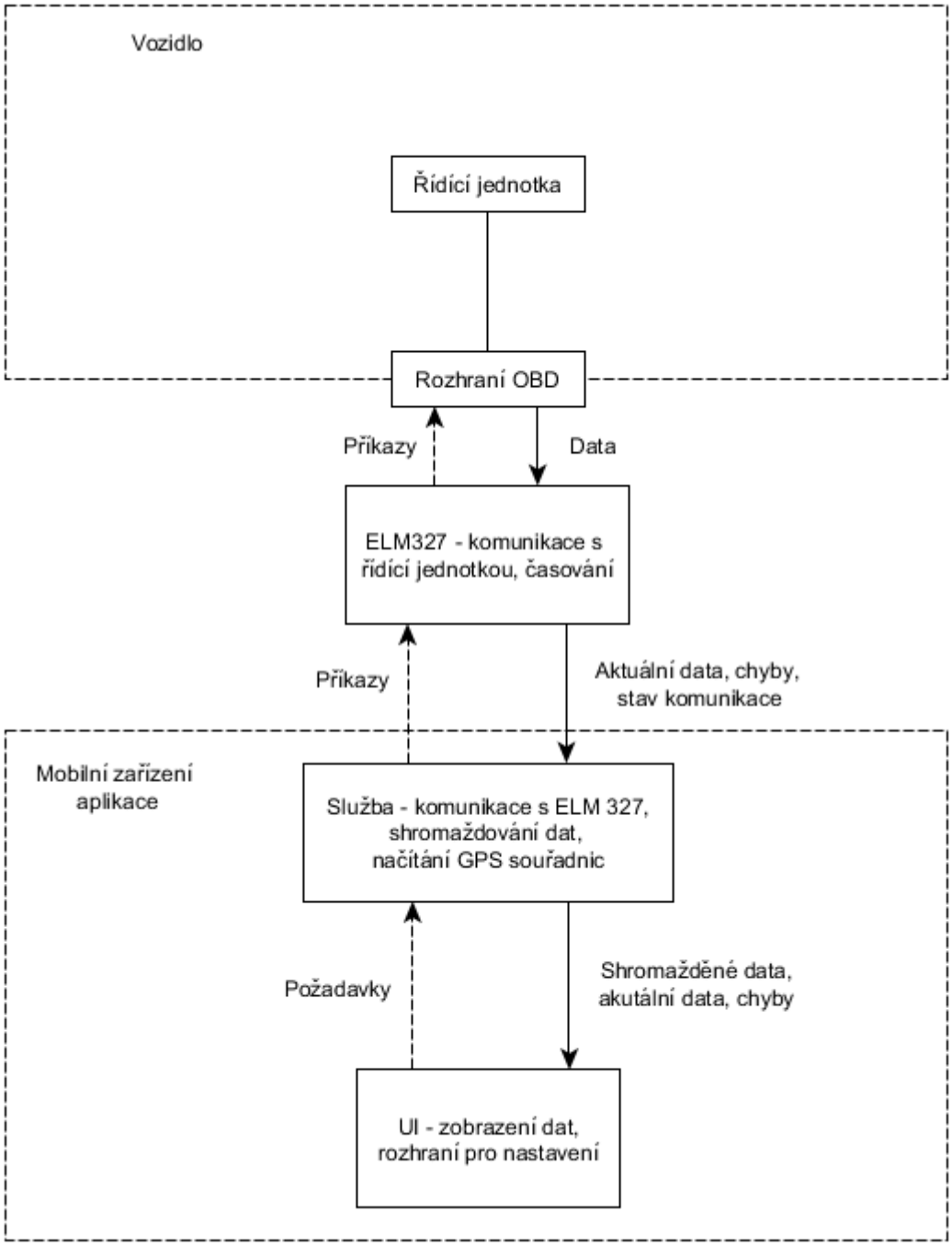
5 Aplikace pro mobilní zařízení

V mobilním zařízení měla být původně pouze aplikace pro uživatelské rozhraní, nicméně logika komunikace byla přesunuta do mobilního zařízení.

Požadavky na fungování mezi uživatelským rozhraním a aplikaci pro komunikaci jsou následující:

- Nezávislost, tedy komunikace musí fungovat i bez spuštěného uživatelského rozhraní.
- UI reaguje ihned, není zdržováno zatížením komunikace.
- V případě spuštění UI se spustí i komunikace (pokud neběží).
- Komunikace mezi sebou.
- Pád komunikace nezpůsobí pád UI a naopak.

Z výše daných požadavků je jasné, že aplikace musí běžet separátně. Standardně se při dlouho běžících aplikací používá služba, která je pro takový účel stvořena. Technologie Androidu umožňuje vytvořit službu, která dokáže běžet v nezávislém kontextu a dokonce dokáže takovou službu vytvořit i v rámci jediné aplikace s uživatelským rozhraním s tím, že zdroje jsou oddělené. Na obrázku 16 je vidět nový návrh celé komunikace s řídící jednotkou až po zobrazení v mobilním zařízení.

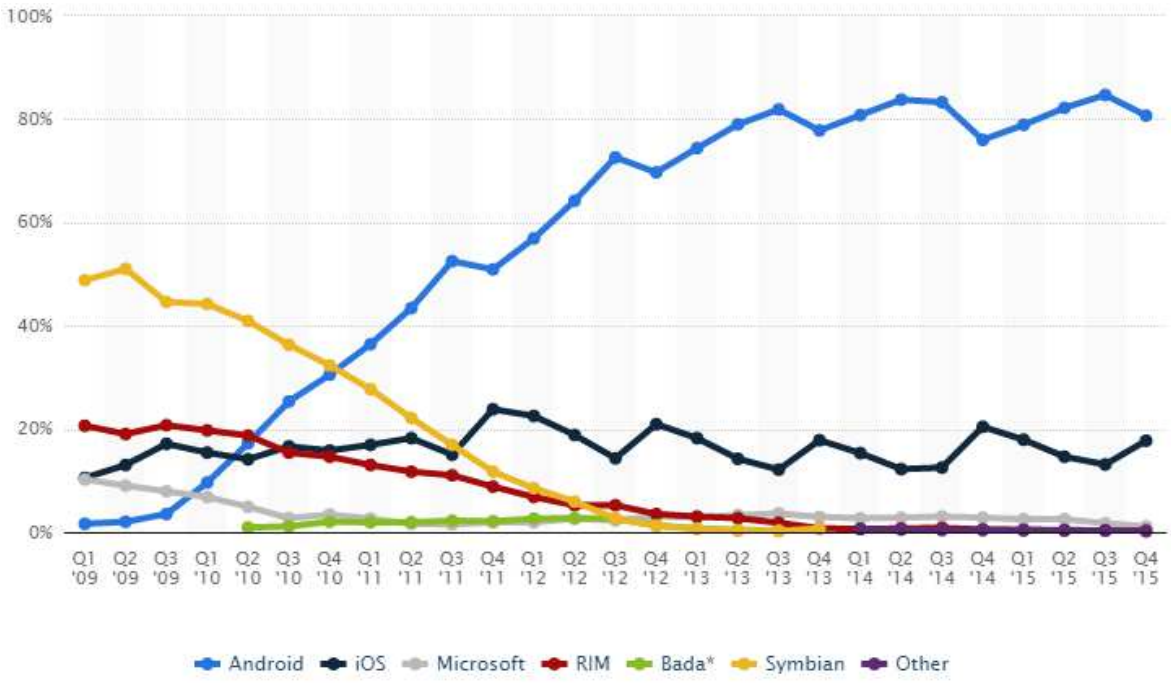


Obrázek 16: Nové rozvržení bez původní mezivrstvy.

5.1 Operační systém Android

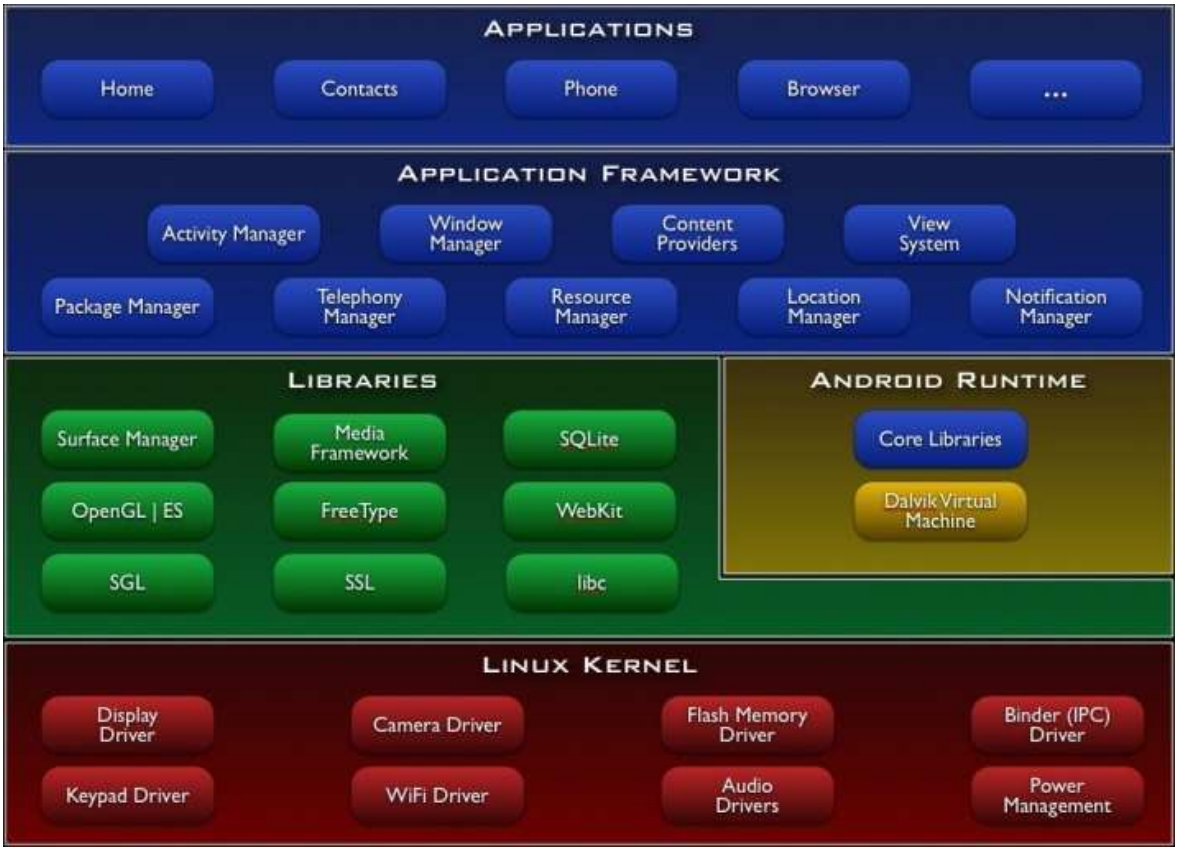
Android je operační systém, který běží na kernelu z Linuxua je určen primárně pro chytré mobilní zařízení a v posledních letech i tablety. Vývoj této platformy započala společnost Android Inc. v roce 2003, která byla v roce 2005 odkoupena světovým gigantem Google. Android byl oficiálně

představen v roce 2007 jako open source platforma, tedy jeho zdrojové kódy jsou volně dostupné a každý je může používat a modifikovat dle vlastního uvážení. Díky této strategii byl Android na konci roku 2010 nejprodávanějším operačním systémem chytrých telefonů a jeho zastoupení na trhu se za poslední čtyři roky drží okolo 80 %. Vývoj zastoupení na trhu můžete vidět na obrázku 17.



Obrázek 17: Graf prodejnosti telefonů dle operačních systémů v čase [13].

Základ Androidu tvoří jádro z Linuxu, které je mezivrstvou mezi softwarem a hardwarem, jenž poskytuje základní funkčnosti jako je správa paměti, správa procesů, řízení práv, řízení spotřeby nebo správu ovladačů. Vedle toho jsou pak přidány další knihovny OpenGL pro práci s grafikou, SQLLite pro práci s databázemi nebo SurfaceManager jenž se stará o zobrazení aplikací a další. Tyto části jsou psány v programovacím jazyce C a C++. Nadstavbou je pak aplikační framework, který poskytuje aplikacím rozhraní pro prácí s prvky operačního systému, např. práce s grafickými komponentami. Jelikož jsou aplikace včetně frameworku psané v jazyce Java, musí nad kernelem běžet komponenty Android Runtime a Dalvik Virtual Machine, které se starají o překlad jazyka do nativního kódu. Náhled na rozvrstvení můžete vidět na obrázku 18.



Obrázek 18: Vrstvy platformy Android [14].

5.2 Služba

Služba nebo služba na pozadí je aplikace, která běží na pozadí operačního systému. Uživatel pak ani neví, že na jeho zařízení taková aplikace běží.

Využívá se k dlouhodobé činnosti, např. sleduje, které klávesy mačkáme, a poté se tyto informace posílají dále na internet. Náročnost služby a její prostředky můžou být omezeny, například pokud aplikace využívá příliš mnoho paměti, pak ji operační systém ukončí nebo restartuje.

Služba s aplikací komunikuje pomocí zpráv, ve kterých může posílat různé data, případné objekty pokud se nejedná o službu na pozadí.

V Androidu se služby dále dělí:

- **Krátkodobá služba** – taková služba vykoná určitou sekvenci příkazů a poté se ukončí.
- **Dlouhodobá služba** – výše uvedená (již popisovaná) služba, která běží neomezenou dobu, ale musí být nenáročná.

Dále se dělí dle kontextu prostředků:

- **Služba v popředí** – běží ve stejném kontextu jako aplikace, pokud havaruje aplikace, pak i služba a naopak. Výhodou je sdílená paměť, takže si služba s aplikací může předávat objekty.
- **Služba na pozadí** – běží nezávisle, může běžet bez uživatelského rozhraní, má oddělené prostředky, v tomto případě si nemůže s aplikací předávat objekty, ale může předávat pouze primitivní typy.

5.2.1 Požadavky na službu

Požadavky na komunikaci a práci s daty byla popsána v kapitole 4. Tato část je zaměřena na zbylé části, které služba musí umět.

- Použití databáze pro ukládání dat.
- Možnost definice vlastních pohledů – uživatel si definuje různé pohledy, kterých může být několik a v každém pohledu můžou být různé ukazatelé, různých veličin. Služba pak musí přizpůsobit komunikaci dle nastavených pohledů a aktuálně prohlížen.
- Nenáročnost na paměť a výkon v době neaktivity vozidla.
- Služba se zapíná automaticky se zapnutím přístroje.

5.2.2 Databáze SQL Lite, ORM lite

Android nativně podporuje databází SQL Lite, která podporuje téměř všechny funkce, které by měla SQL databáze podporovat, takže pro navrhovanou aplikaci je naprosto dostačující. Hlavní slabinou této malé databáze je rychlost, která není nijak ohromující, nicméně pro naše použití by měla být dostačující i při záznamech v řádu statisíců.

Základní příkazy pro manipulaci s daty a pro definici dat jsou totožné nebo velice podobné jak u databází MS SQL nebo Oracle SQL, takže se nebudu zabývat ani její syntaxi.

Zajímavější je mapování databáze pomocí ORM Lite. Jedná se o knihovnu třetí strany, která dokáže mapovat třídy, její atributy a vazby. Pracuje s objekty pomoci repositáře, který na požádání vytváří a maže tabulky či ukládá, načítá a maže záznamy. Ukázku pro získání repositáře můžete vidět v ukázce kódu 4, v ukázce kódu 5 a 6 pak najdete příklad pro vytvoření a smazání tabulky a v ukázce 7 a 8 můžete vidět uložení a načtení záznamu z tabulky.

Kompletní ER diagram navržené aplikace je vidět na obrázku 21.

```
public RuntimeExceptionDao<ObdCommandDefinition, Integer>
getGaugePositionDefinitionRepository() {
    if (gaugePositionDefinitionRepository == null) {
        gaugePositionDefinitionRepository =
getRuntimeExceptionDao(ObdCommandDefinition.class);
    }
    return gaugePositionDefinitionRepository;
}
```

Ukázka kódu 4: Získání repositáře pro komunikaci s databází.

```
TableUtils.createTable(DatabaseConnection.DatabaseHelper.getConnectionSource(),
ObdCommandDefinition.class);
```

Ukázka kódu 5: Vytvoření tabulky v databázi.

```
TableUtils.dropTable(DatabaseConnection.DatabaseHelper.getConnectionSource(),
ObdCommandDefinition.class,true);
```

Ukázka kódu 6: Smazání tabulky z databáze.

```
getObdCommandDefinitionRepository().createOrUpdate((ObdCommandDefinition)
entity);
```

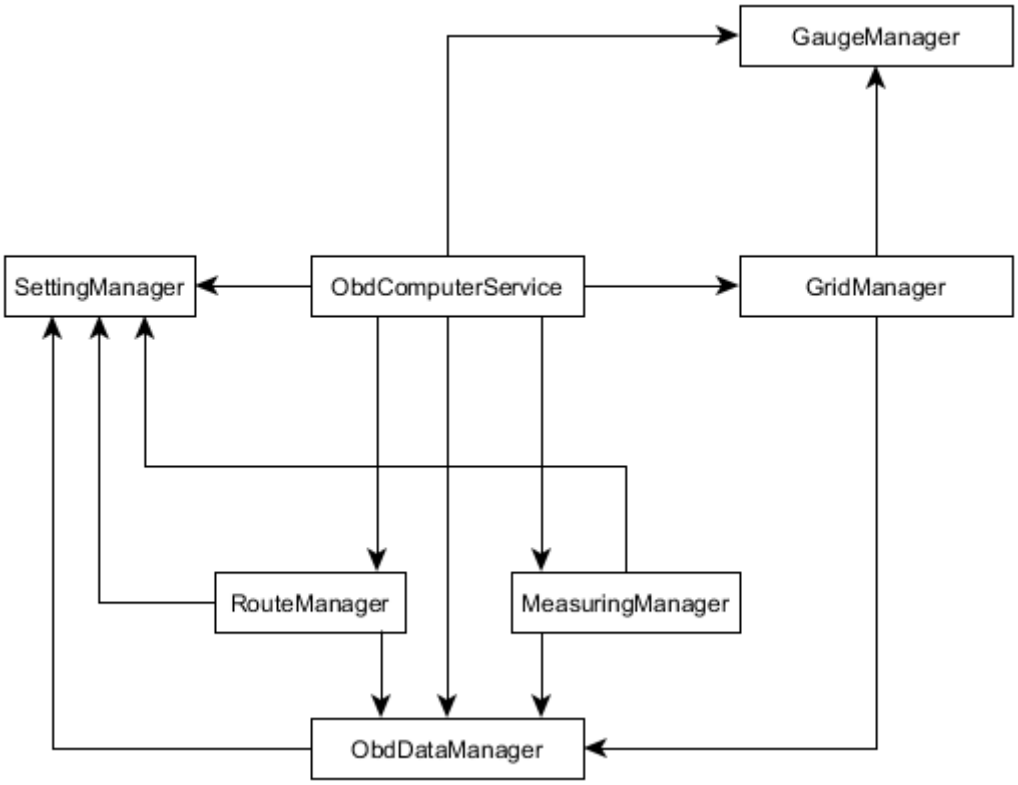
Ukázka kódu 7: Uložení dat do databáze.

```
obdCommandDefinitionList =
DatabaseConnection.DatabaseHelper.getObdCommandDefinitionRepository().
queryForAll();
```

Ukázka kódu 8: Načítání dat z databáze.

5.3 Implementace služby

V minulé kapitole byla popsána část implementace pro komunikaci, která je ve službě použitá a je rozšířená o další požadavky. Služba obsluhuje 6 komponent, některé komponenty byly popsány již výše, ostatní jsou popsaný dále.



Obrázek 19: Rozložení komponent ve službě.

5.3.1 ObdComputerService – služba

Komponentu tvoří třída, která dědí z třídy „Service“ a implementuje některé její metody, tím se stává službou. Jedná se o službu, která běží na pozadí a registruje se při spuštění zařízení.

Při prvotním zavolání služby, neboli při vytvoření služby, se inicializují všechny komponenty znázorněné na obrázku 19. Poté služba naslouchá požadavkům přicházejících z UI a pomocí zpráv odpovídá. Obsahem zprávy můžou být pouze primitivní typy, takže je potřeba veškeré objekty převést do primitivních typů, více v kapitole 5.3.5.

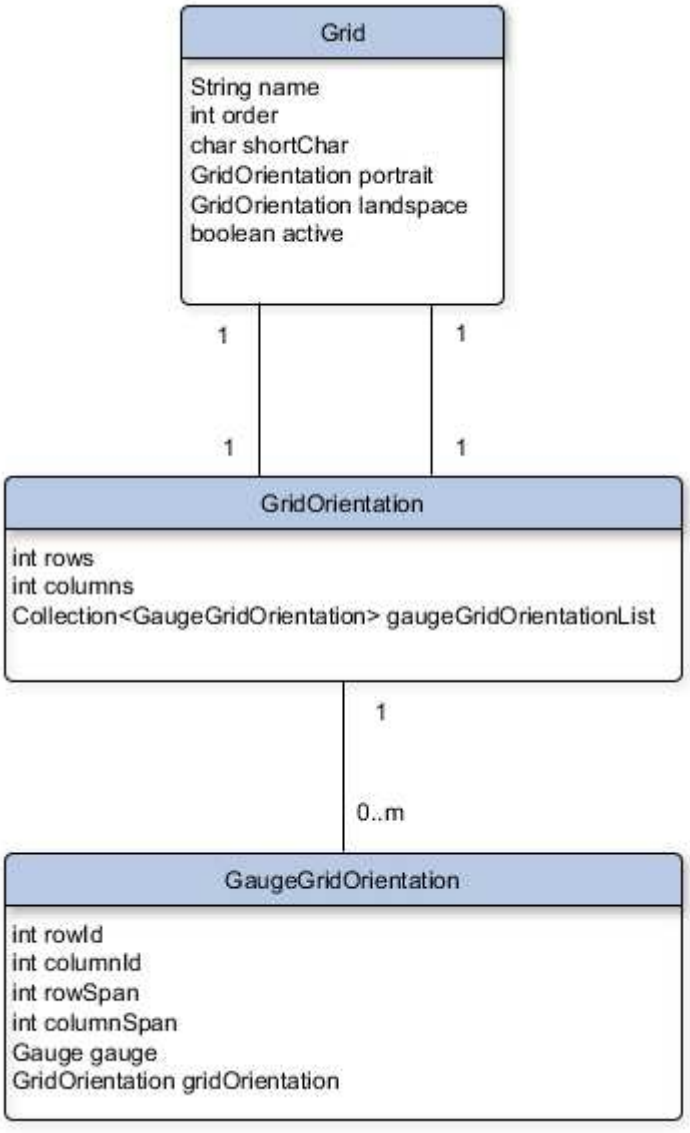
5.3.2 GaugeManager

Komponenta spravuje ukazatele a jejich definice. Ukazatelem se myslí zobrazení, např. tachometr, které má několik míst pro zobrazení hodnot a veličin. V aplikaci je nadefinovaných několik

zobrazení a u každého jsou zobrazeny pozice, které by měly být naplněné zdrojem dat. Zdrojem dat jsou čítače popsané výše v kapitole 4.4.3. V rámci ukazatele lze použít pro každou pozici jiný čítač.

5.3.3 GridManager

Řeší správu mřížek, které si uživatel může prostřednictvím UI definovat. V rámci nastavení si může definovat neomezený počet mřížek. Každá mřížka se definuje dvakrát, jedno zobrazení je na výšku (portrait) a druhé na šířku (landspace). V rámci mřížky si může definovat počet řádku a sloupců. V každé buňce pak může přidat jeden ukazatel, kterému může nastavit zobrazení přes několik řádků nebo sloupců mřížky. Každá mřížka má svůj název, zkratku a pořadí.



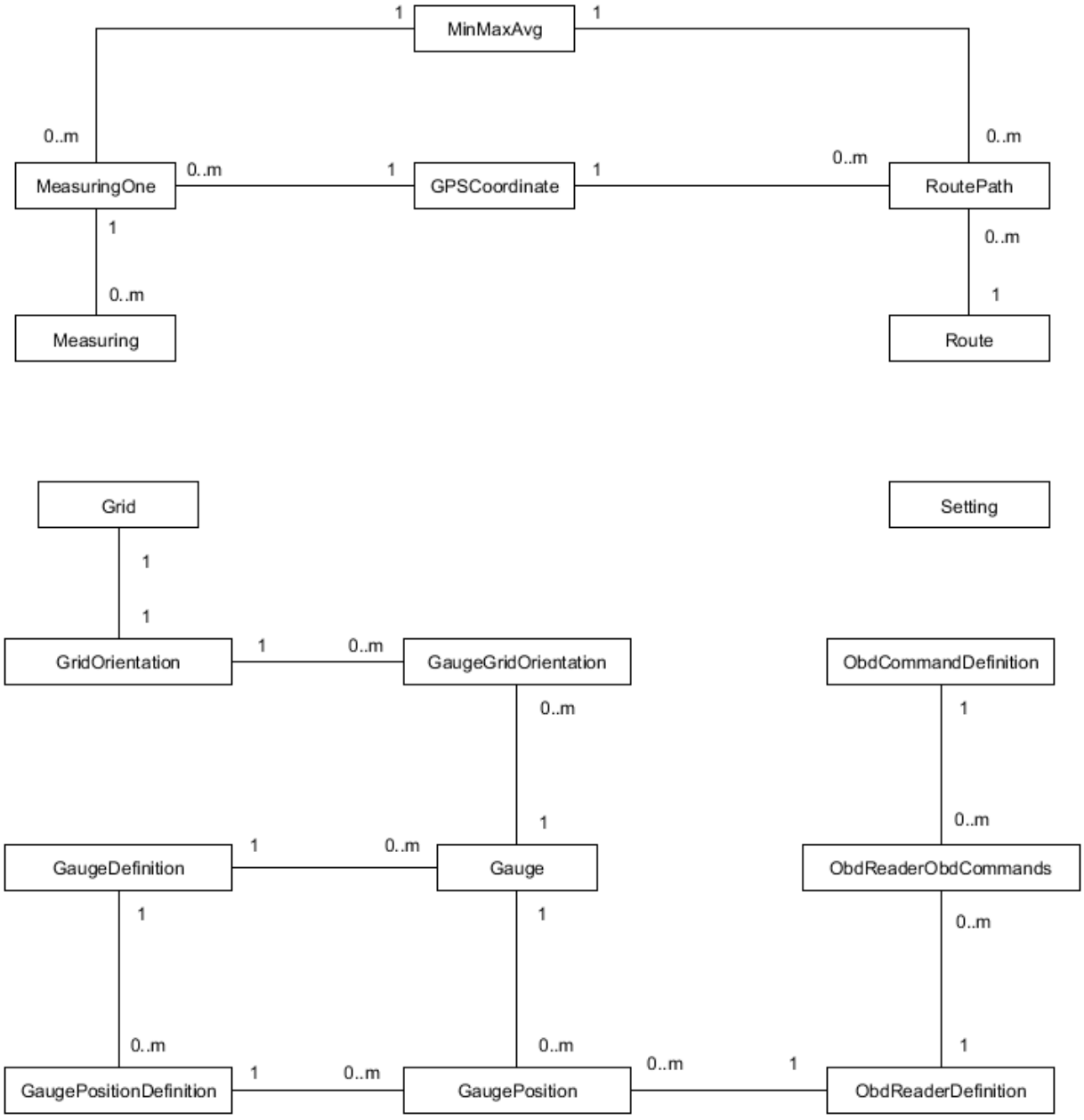
Obrázek 20: Třídní diagram tříd Grid, GridOrientation a GaugeGridOrientation.

5.3.4 SettingManager

Manažer nastavení umožňuje prostřednictvím UI určit některé hodnoty. Tyto hodnoty si poté načítají ostatní komponenty. Komponenty si můžou hodnoty buď načítat napřímo, nebo se můžou registrovat a odposlouchávat změny.

Nastavitelné hodnoty:

- Bluetooth zařízení.
- ELM327 OBD protokol.
- Simulace ECU.
- Ukládání dat ze simulace.
- Maximální délka úseku.
- Maximální čas úseku.
- Orientace obrazovky.



Obrázek 21: Kompletní rozvržení databáze ve službě.

5.3.5 Komunikace mezi službou a UI

Služba komunikuje prostřednictví „broadcastu“, komunikace je obousměrná, tedy jak UI, tak služba může kdykoliv poslat požadavek, častěji se doptává UI a služba odpovídá. Služba může odpovědět, aniž by se UI doptávalo.

```
public int onStartCommand(final Intent intent, int flags, int startId) {
if ( intent != null && intent.getStringExtra("ACTION") != null ) {
String actionText = intent.getStringExtra("ACTION").toString();
ObdComputerServiceReceiver.Action action =
ObdComputerServiceReceiver.Action.valueOf(actionText);

Intent broadcastIntent = new Intent();
broadcastIntent.setAction(ObdComputerServiceReceiver.PROCESS_RESPONSE);
broadcastIntent.addCategory(Intent.CATEGORY_DEFAULT);
broadcastIntent.putExtra("ACTION",action.toString());
switch (action)
            {

case GetAllRoutes:{
ArrayList<Route> routes = (ArrayList<Route>)routeManager.GetRouteList();
broadcastIntent.putParcelableArrayListExtra("DATA", routes);
break;
}
                …
            }
thisObject.sendBroadcast(broadcastIntent);
return START_STICKY;
}
```

Ukázka kódu 9: Zpracování požadavku ve službě.

Příkazem „sendBroadcast“ pošleme odpověď, v tomto případě seznam provedených jízd.

Pro získání odpovědi pak potřebujeme vytvořit třídu, která dědí z třídy „BroadcastReceiver“. Tato třída pak naslouchá zaslaným zprávám, které může posléze zpracovat.

```
public class ObdComputerServiceReceiver extends BroadcastReceiver {

public static final String ACTION_GET_DATA_MANAGER = "GET_DATA_MANAGER";
public static final String PROCESS_RESPONSE =
"onboardcomputer.computer.ObdComputerStartReceiver";

private IObdComputerServiceListener obdComputerServiceListener;

public ObdComputerServiceReceiver(IObdComputerServiceListener
obdComputerServiceListener) {
this.obdComputerServiceListener = obdComputerServiceListener;
}

@Override
public void onReceive(Context context, Intent intent) {

Action action = Action.valueOf(intent.getStringExtra("ACTION"));

switch ( action ) {
case GetAllRoutes:{
                List<Route> routes =
intent.getParcelableArrayListExtra("DATA");
obdComputerServiceListener.OnReceiveAllRoutes(routes);
break;
}
```

Ukázka kódu 10: Zpracování odpovědi služby.

Jak již bylo dříve zmíněno, pokud služba běží na pozadí, běží v odděleném kontextu a nesdílí paměť s UI. Tím pádem nemůžeme předávat referenční typy (odkazem do paměti).

Z tohoto důvodu u všech přenášených objektů byla provedena implementace rozhraní „parcelable“, které umožňuje objekt rozebrat na základní primitivní typy. Tím jsme schopni posílat celé objekty, aniž by si aplikace přistupovaly do paměti. Rozhraní „parcelable“ pak automaticky poskládá data složené z primitivních typů dohromady a vytvoří tak nový objekt. Do rozhraní „parcelable“ ukládáme pouze informace, které požadujeme, takže nemusíme posílat veškeré údaje a tím komunikaci ulehčíme. Pomocí tohoto principu jsme schopni posílat i seznamy objektů, nebo objekty v objektu. U druhého zmiňovaného principu hrozí nebezpečí zacyklení, takže je potřeba uvědomit si, z které strany budou objekty na sebe odkazovat.

```
public class GaugePosition extends Entity implements Parcelable {
public GaugePosition(Parcel in)
    {
id = in.readInt();
positionName = in.readString();
obdReaderSystemName = in.readString();
unit = in.readString();
String enumType = in.readString();
if ( enumType != null && enumType.length() >0 ) {
dataType = DataTypeEnum.valueOf(enumType);
}
minValue = in.readFloat();
maxValue = in.readFloat();
currentValue = in.readDouble();
this.storeDataId = in.readInt();
String enumStoreType = in.readString();
if ( enumStoreType != null && enumStoreType.length() >0 )  {
storeType = StoreTypeEnum.valueOf(enumStoreType);
}
    }
@Override
public int describeContents() {
return 0;
}
@Override
public void writeToParcel(Parcel dest, int flags) {
        dest.writeInt(id);
dest.writeString(positionName);
dest.writeString(obdReaderSystemName);
dest.writeString(unit);
dest.writeString(dataType.toString());
dest.writeFloat(minValue);
dest.writeFloat(maxValue);
dest.writeDouble(currentValue);
dest.writeInt(storeDataId);
dest.writeString(storeType.toString());
}
public static final Parcelable.Creator CREATOR = new Parcelable.Creator() {
public GaugePosition createFromParcel(Parcel in) {
return new GaugePosition(in);
}
public GaugePosition[] newArray(int size) {
return new GaugePosition[size];
}
    };
```

Ukázka kódu 11: Použití „parcelable“.

Ukázka použití „parcelable“, třída nejprve musí dědit z rozhraní „parcelable“, poté musí implementovat metody „writeToParcel“, „describeContents“ a konstruktor s parametrem „Parcel in“. Pokud se daný objekt používá i v poli, pak musíme implementaci rozšířit o „Parcelable.Creator“, tedy předpis jak se má pole tvořit.

5.4 Implementace uživatelského rozhraní

Pro zobrazení získaných údajů potřebujeme rozhraní, grafické uživatelské rozhraní, které bude pro uživatele přívětivé. Android umožňuje tvořit základní uživatelské rozhraní pomocí aktivit.

Aktivita se skládá minimálně z třídy a xml souboru, který definuje vzhled aktivity. Vzhled aktivity se skládá pomocí dostupných „View“, tedy zobrazení.

Zobrazení jsou rozděleny do několika sekcí:

- Layouty.
- Widgety.
- Textové pole.
- Kontejnery.
- Pole s datem a časem.
- Pokročilé zobrazení.
- Vlastní zobrazení.

Mezi základní zobrazení patří layouty, widgety a textová pole.

Layout je rozdělení zobrazení do několika segmentů. V Androidu existuje několik typů a to lineární horizontální, lineární vertikální, tabulkové zobrazení, řádek tabulky, mřížka, relativní a rámový layout.

Widgety jsou systémové zobrazení, např. pole s textem nebo tlačítka a další.

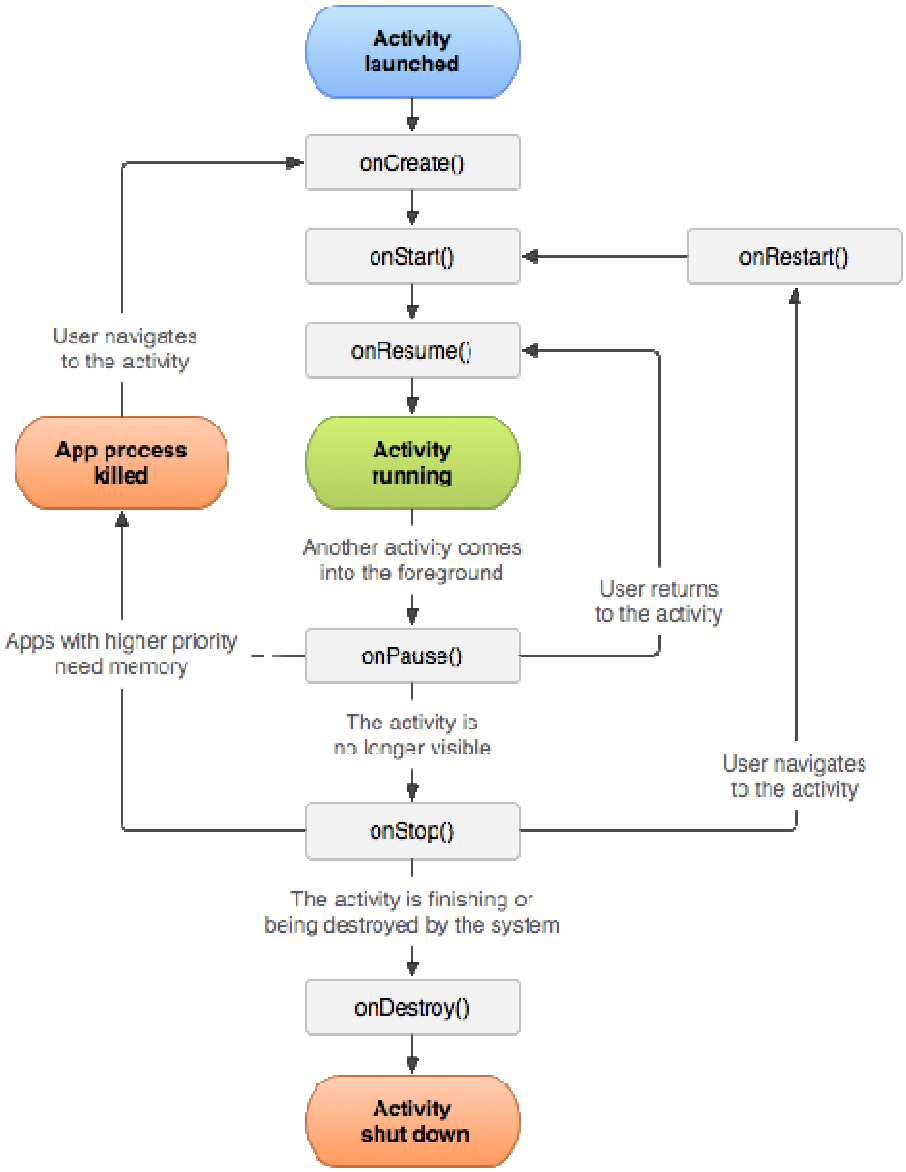
```
        <?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical" android:layout_width="match_parent"
android:layout_height="match_parent">

<include
android:id="@+id/toolbar"
layout="@layout/toolbar" />


<GridLayout
android:layout_width="match_parent"
android:layout_height="match_parent"
android:id="@+id/glMainGrid"></GridLayout>
</LinearLayout>
```

Ukázka kódu 12: Definice vzhledu aktivity pomocí XML.

Druhá část je třída, která dědí z třídy „Activity“ případně dalších tříd, které z ní dědí a obsahuje určité metody. Třída nejdříve nastaví zobrazení, se kterým bude pracovat a poté může pracovat s jejími objekty a zobrazeními.

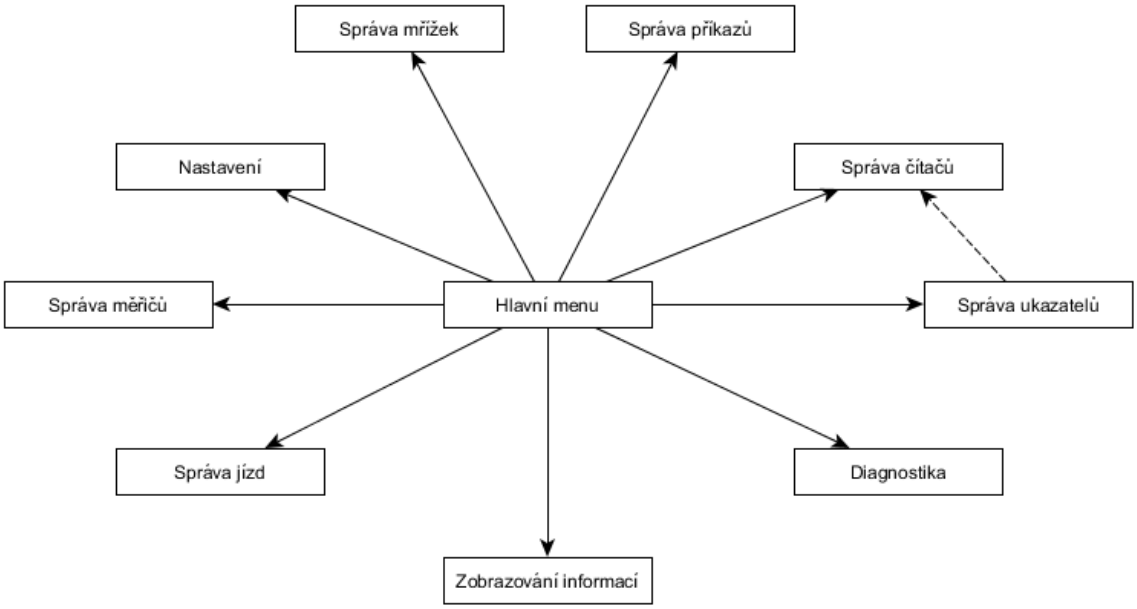
```
public class GridActivity extends ServiceActivity  {
private List<Grid>gridList = new ArrayList<Grid>();
private GridLayout glMainGrid;

public GridActivity()
    {

    }

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

setContentView(R.layout.grid);

glMainGrid = (GridLayout) findViewById(R.id.glMainGrid);

        …
    }
    …
}
```

Ukázka kódu 13: Třída, která se stává aktivitou.



Obrázek 22: Životní cyklus aktivity [15].

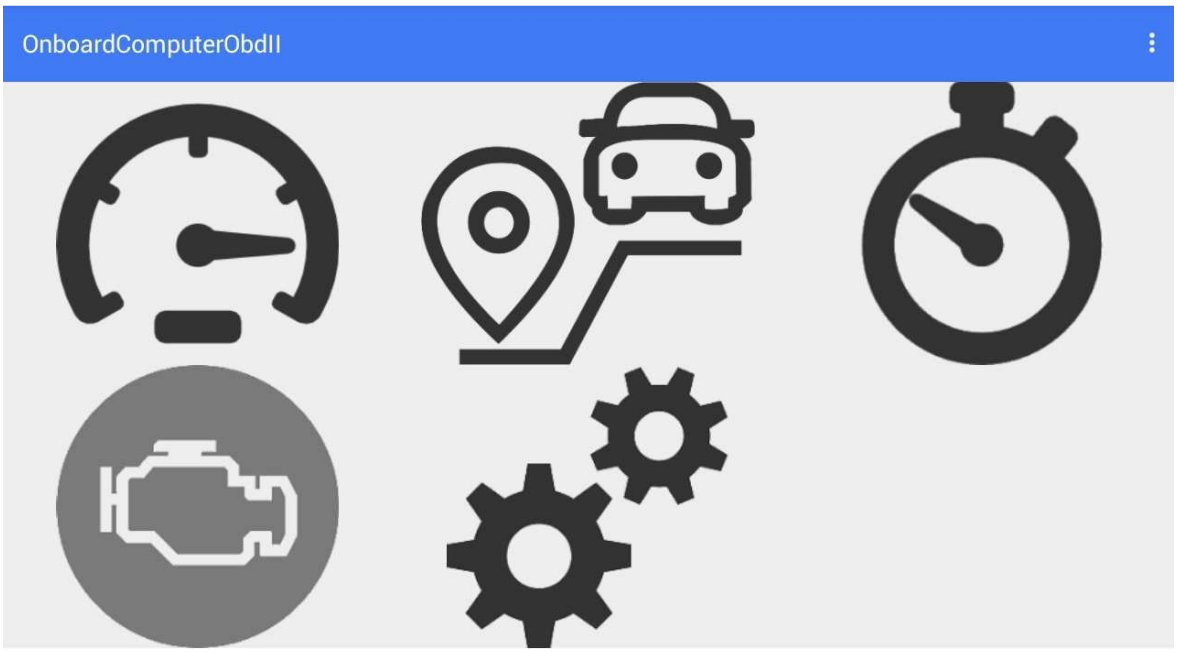
Uživatelské rozhraní se skládá celkem z deseti částí, kde jsou jednotlivé části propojeny z hlavní aktivity, jak je zřejmé z obrázku 23. Hlavní aktivita je tedy rozcestník do všech ostatních částí rozhraní.



Obrázek 23: Rozvržení uživatelského rozhraní.

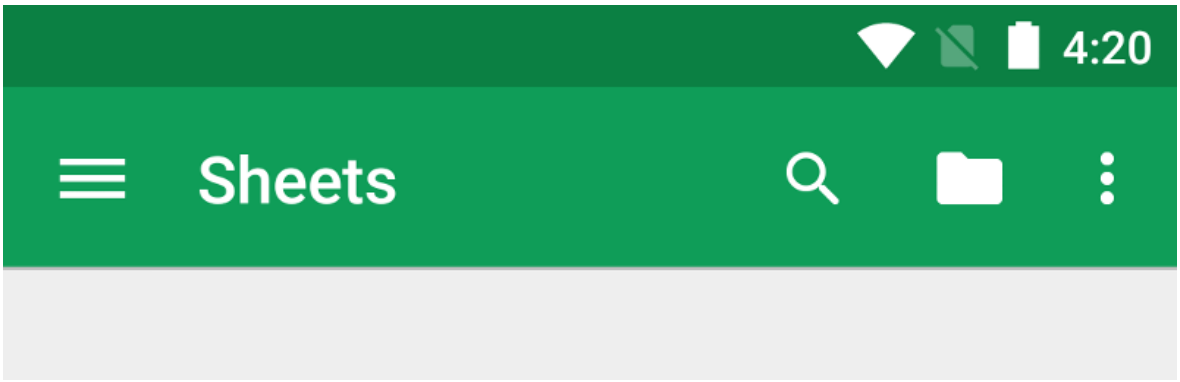
5.4.1 Hlavní aktivita

Hlavní aktivita je složena z navigační lišty a 5ti obrázků, které slouží jako odkazy do dalších částí rozhraní.



Obrázek 24: Ukázka rozcestníku z aplikace.

Navigační lišta je tvořena pomocí Android komponenty „ActionBar“, která umožňuje vytvořit panel uchycený na hranu obrazovky, kde jsme schopni vypisovat texty, tlačítka, ikonky nebo menu, jež zobrazí další odkazy, které se do panelu nevlezly.



Obrázek 25: Ukázka komponenty „ActionBar“ [16].

5.4.2 Správa příkazů

Ve správě příkazů můžeme přidávat, mazat nebo měnit již existující příkazy. Existující příkazy jsou zobrazené v listu, po zmáčknutí řádku listu nebo tlačítka vyskočí dialogové okno, kde můžeme definovat tyto vlastnosti:

- Systémový název.
- Název.
- Obd kód.
- Počet navracejících se bajtů.
- Popis.
- Jednotka.
- Funkce (rovnice, pomocí které vypočteme smysluplné hodnoty).

Funkce se zapisuje v syntaxi javascriptu, kde jednotlivé bajty jsou uložené v proměnných a, b, c, případně dalších.

Dále zde najdeme informaci, zdali je příkaz systémový nebo o dostupnosti příkazů z posledního připojení řídící jednotky. Pokud definujeme nový příkaz, pak se dostupnost nastaví až po prvním připojení.

Obrázek 26: Ukázka některých příkazů, dostupné jsou podbarveny zeleně.

5.4.3 Správa čítačů

Správa čítačů je obdoba správě příkazů. Můžeme zde definovat nové nebo stávající čítače. Systémových čítačů existuje pouze několik, ostatní si uživatel musí dodefinovat sám, případně se vytvoří automaticky ve správě ukazatelů.

Ve vyskakovacím okně můžeme definovat:

- Systémový název.
- Název.
- Jednotku.
- Příznak pro neustále vykonávaní příkazu.
- Funkci.
- Celkovou funkci.

Funkce jsou opět v syntaxi javacriptu a při psaní funkce máme v dialogovém okně dostupný list všech příkazů, který můžeme v čítači použít. Po uložení se z funkcí vyčtou veškeré příkazy, u kterých poté můžeme definovat prioritu. Význam priority byl popsán výše v kapitole 4.4.3.

Dále zde najdeme informaci o tom, zdali se jedná o systémem vytvořený čítač nebo je vytvořen uživatelem.

5.4.4 Správa měřičů

Zde si můžeme definovat vlastní měřiče, které můžeme kdykoliv vynulovat a tím si můžeme měřit různé úseky. Měřič je totožný s měřidly ujeté vzdálenosti ve vozidle, které můžeme nulovat. Ve správě si však můžeme definovat nekonečný počet měřičů, které můžeme nezávisle nulovat.

V listu vidíme všechny měřice se základními informacemi jako je název, poslední datum a čas vynulování, ujetá vzdálenost, průměrná spotřeba, celková spotřeba a doba jízdy.

Po zobrazení konkrétního měřiče můžeme měřič pozastavit, spustit nebo ho vynulovat. Dále zde vidíme historii úseků mezi nulováním. Kromě základních informací, které jsou dostupné i v listu (pouze pro poslední záznam), zde najdeme i informace z čítačů, které v době daného úseku měly příznak, že se mají neustále měřit.

Idea je možnost nastavit i automatické nulování, například při tankování, nebo automatické pozastavení nebo znovu spuštění, ale tato možnost prozatím nebyla implementována.

5.4.5 Správa jízd

Správa jízd je obdoba správě měřičů, opět můžeme vidět seznam všech provedených jízd, kde jsou základní informace, jako je datum odjezdu, datum dojezdu, čas jízdy, průměrná spotřeba, celková spotřeba, průměrná rychlost a ujetá vzdálenost. V listu pak můžeme omezit výběr dle data od - do.

Po zobrazení konkrétní jízdy dále vidíme GPS souřadnice odjezdu a dojezdu (pokud byl GPS modul dostupný) nebo data z ostatních čítačů, které měli v době jízdy nastaveny příznak pro neustálé měření.

Bohužel prozatím nebyla zapracována možnost zobrazení mapy se zobrazením trasy, nicméně informace o průběhu trasy se ukládají.

Ty můžeme najít v další částí, kde najdeme základní informace o jednotlivých částech trasy. Části trasy se rozumí úsek, který je maximálně dlouhý např. 100 metrů nebo maximální čas 10 sekund. Tyto údaje se volí v nastavení.

O jednotlivých úsecích jízdy najdeme tyto informace:

- GPS souřadnice.
- Datum a čas vytvoření úseku.
- Čas.
- Spotřeba.
- Průměrná rychlost.
- Průměrná spotřeba.

Z těchto informací si taky můžete zobrazit graf.

5.4.6 Správa ukazatelů

Ukazatele jsou grafické zobrazení, které mají za úkol zobrazovat definované informace. Zobrazení jsou už předem definované, a ve správě ukazatelů můžeme vytvářet jednotlivé instance a přiřazovat jim zdroje dat, včetně pozic a dalších popisků. Jeden z ukazatelů je vidět na obrázku 27.

Obrázek 27: Ukazatel z aplikace.

První část správy ukazatelů tvoří list, kde můžeme vidět veškeré definované instance ukazatelů. Kromě názvu v listu vidíme vždy prefix, který označuje druh instance například „TachometerView“. Na list můžeme zmáčknout a tím můžeme editovat nebo smazat danou instanci. Kromě této možnosti je níže dostupné tlačítko pro vytvoření nové instance.

Druhá část je dialogové okno, kde si můžeme definovat název, druh instance. Dále zde najdeme definované pozice, případně můžeme přidat další, pokud jsme již všechny pozice nevyužili. Po zmáčknutí na definovanou pozici se zobrazí další dialogové okno.

Třetí část je druhé dialogové okno, kde můžeme definovat zdroj a cíl zobrazení a další informace.

Jako první definujeme cíl zobrazení. Cíle můžou být různé dle vybraného typu zobrazení. Na obrázku 28 jsou znázorněny tři pozice, které se definují pro daný ukazatel. Další část je zdroj, tedy čítač, ze kterého se budou data načítat. Ve výběru vidíme všechny definované čítače a příkazy. Pokud zmáčkneme na příkaz, pak se automaticky z daného příkazu vytvoří čítač, který je použitý jako zdroj dat. Se zdrojem pak souvisí další položky, kde volíme typ dat. Typem dat se myslí, zdali se použije průměrná, minimální, maximální, celková nebo aktuální hodnota. Pak můžeme zvolit, zdali se data budou načítat z aktuální jízdy, minulé jízdy, z některého z měřičů nebo od zapnutí aplikace. Pokud zvolíme, že se data budou načítat z měřice, pak si v další kolonce musíme zvolit, který měřič bude použit.

Obrázek 28: Ukazatel s vyznačenými pozicemi.

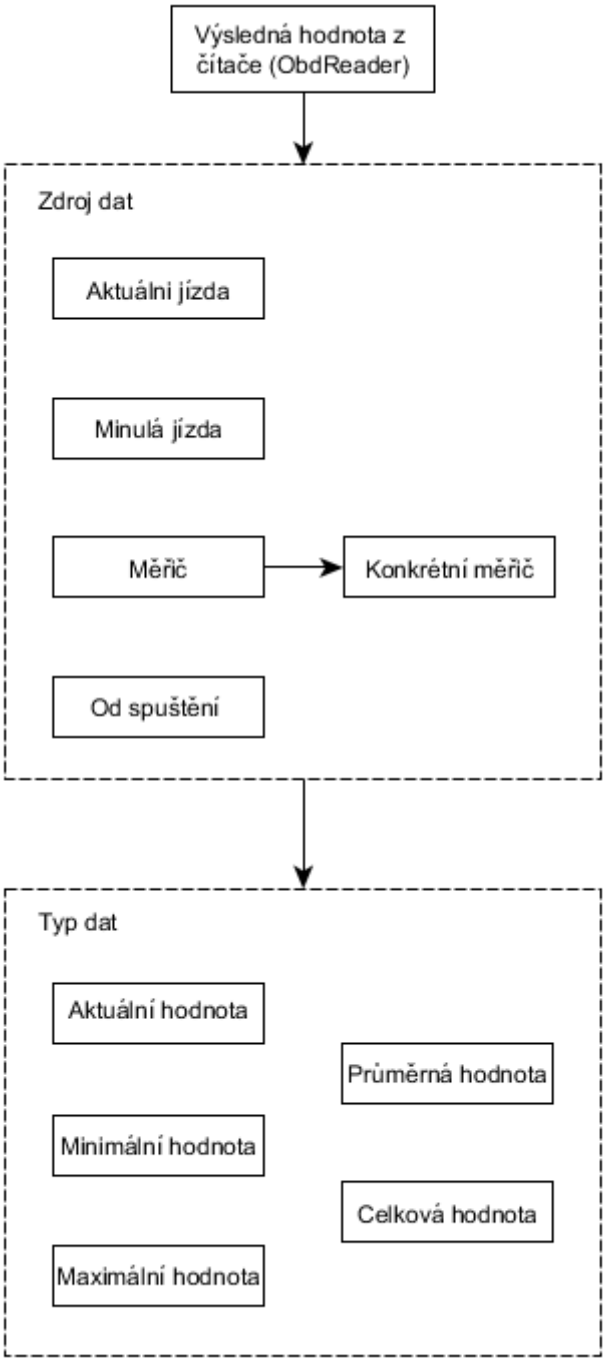
Poté definujeme ostatní informace dané pozice ukazatele jako je:

- Jednotka.
- Minimální hodnota.
- Maximální hodnota.

Minimální a maximální hodnotou se myslí, do jakých krajních bodů může daný ukazatel informace zobrazovat. Pro představu se jedná o krajní body tachometru v automobilu, kde minimální hodnota je 0 a maximální hodnota 200 km/h. Typický příklad kdy je tuto možnost vhodné využít je aktuální spotřeba v litrech na sto kilometrů, protože když se rychlost auta blíží k nule, pak se hodnota spotřeby přibližuje k nekonečnu. Tímto omezíme zobrazení na maximálně 50 l/100 km, protože v těchto hodnotách se spotřeba pohybuje nejčastěji.

V případě, že minimální a maximální hodnotu necháme nulovou, pak se krajní hodnoty budou dynamicky nastavovat dle dosažených hodnot. Toto má smysl hlavně při zobrazení jako je tachometr s ručičkou, při zobrazování prostých čísel se omezení neaplikuje a zobrazí se vždy neomezená hodnota.

Při výběru zdroje, tedy čítače případné příkazu, se automaticky nastaví jednotka, tu ale můžeme změnit.



Obrázek 29: Názorná ukázka výběru dat pro ukazatele.

5.4.7 Správa mřížek

Poté co si definujete ukazatele, můžete ukazatele použít v mřížkách. V mřížce si definujete *x* sloupců a *y* řádku. Tím vznikne *x* krát *y* buněk, kde do každé buňky můžete vložit nějaký ukazatel, případně ho můžete nechat prázdný. Ukazatel taky můžete zobrazit přes několik řádku nebo sloupců a tím bude ukazatel větší než ostatní.

Takto si můžeme definovat několik mřížek, které si pak můžeme přepínat v aktuálně zobrazovaných datech. Každá mřížka má název, pořadí, zkratku a příznak zdali je aktivní. Zkratka se využívá pro rychlejší průchod mřížek při provozu.

Každá mřížka se definuje 2×, jednou pro zobrazení horizontálně a podruhé pro vertikální zobrazení.

5.4.8 Nastavení

Každá aplikace poskytuje nějaké rozhraní pro nastavení proměnných. V této části si můžeme nastavit tyto vlastnosti:

- Bluetooth zařízení.
- ELM327 protokol.
- Simulaci provozu.
- Ukládání dat ze simulace provozu.
- Maximální vzdálenost úseku jízdy.
- Maximální čas jednoho úseku jízdy.
- Orientaci obrazovky.

U protokolu ELM327 můžeme vybírat ze všech dostupných protokolů, viz kapitola 4.3, dále můžeme zvolit automatické hledání, které podporuje ELM327 rozhraní, nebo aplikaci definovaný typ, který upřednostňuje naposled připojený protokol, jinak zkouší připojení i přes všechny ostatní protokoly.

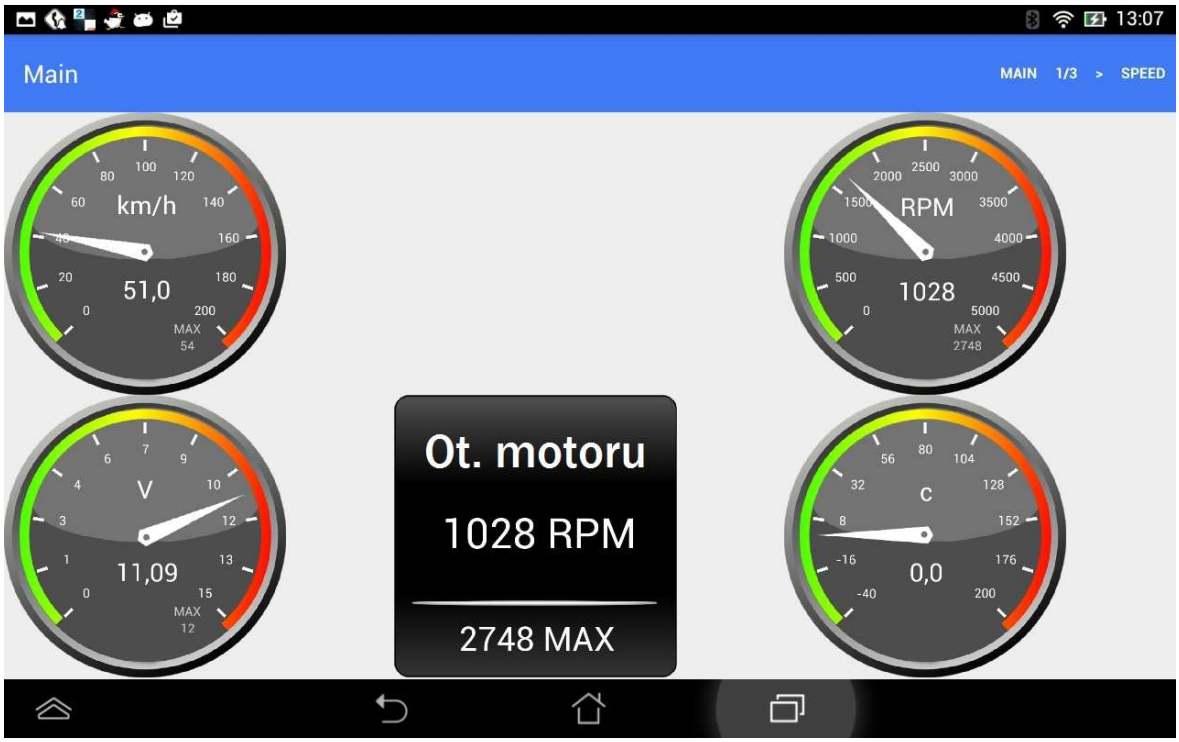
Simulace provozu znamená, že se aplikace tváří, že je připojená k řídicí jednotce a poskytuje některé základní údaje. Uživatel si může vyzkoušet, že jeho vlastní nadefinované ukazatelé fungují správně. Dále je zde možno najít příznak, zdali se data načtená ze simulace budou ukládat.

Maximální vzdálenost a čas úseku jízdy se nastavuje v metrech respektive v sekundách.

Orientace obrazovky znamená, zdali se bude aplikace zobrazovat pouze horizontálně, vertikálně nebo automaticky dle natočení tabletu.

5.4.9 Zobrazování aktuálních informací neboli pracovní plocha

V této části najdeme všechny mřížky, které jsou předefinované v části definic mřížek a jsou aktivní. Kromě ukazatelů vidíme i hodnoty, které jsou načtené z námi definovaných ukazatelů. Pohybovat se mezi mřížkami můžeme pomocí přetažení prstu po tabletu zleva doprava nebo zprava doleva. Zobrazování aktuálních informací z aplikace je vyobrazeno na obrázku 30.



Obrázek 30:Pracovní plocha z aplikace.

5.4.10 Diagnostika

Diagnostika umožňuje vyčíst chybové hlášky případně je vymazat. Kromě těchto informací si můžeme zobrazit i readiness kódy, které se používají nově od roku 2016 v České republice pro diagnostiku emisí [17].

Readiness kódy jsou bity, které odpovídají výsledkům „testů“ z některých komponent vozidla. Nejedná se přímo o testy, ale o průběžné monitorováni funkčnosti a v případě významných odchylek řídící jednotka určí, že daná komponenta nefunguje správně. Jelikož se jedná o bity, je potřeba si nejprve vyčíst, které testy vozidlo podporuje, a poté vyčíst, které z podporovaných testů jsou úspěšné. Způsob výčtu těchto kódů je popsán v kapitole 2.1.3. Nesplněné testy pak znamenají, že daná komponenta nepracuje korektně, anebo že daný test ještě nebyl dokončen. To nastává zejména ve chvíli, kdy jsme promazali chyby řídící jednotky nebo jsme byli s vozidlem v servise, kde takový úkon nejspíše provedli. V praxi to pak znamená, že po zásahu v servise musíme před kontrolou emisí s vozidlem ujet minimálně několik desítek kilometru, u některých vozidel až stovek kilometrů. Pokud i po ujetí cca 500 km zjistíme, že daný test nebyl úspěšně dokončen, pak je komponenta nefunkční. Monitorování probíhá neustále, takže pokud by komponenta z nějakého důvodu po čase začala opět fungovat, pak by se test uložil s příznakem, že je úspěšný a naopak [3].

6 Porovnání měřených hodnot

Největší problém je věrohodnost naměřené spotřeby. Pokud řídicí jednotka neposkytuje konkrétní informaci o aktuální spotřebě, pak můžeme spotřebu pouze dopočítávat z dostupných informací a těžko bude přesně odpovídat realitě. Nicméně ani hodnoty z palubních počítačů nejsou přesné a jediné o co se můžeme opřít je ruční měření spotřeby dle tankování. V kapitole 4.4.3 byly popsány dvě alternativní metody výpočtu aktuální spotřeby, z toho u jedné metody se váha vzduchu dá nahradit pomocí dalších hodnot z jiných čidel, která se pak ideálně násobí poměrem vzduchu a paliva.

Provedl jsem proto test, kde jsem měřil ujetou vzdálenosti a spotřebu paliva z různých zdrojů a výpočtů:

- Rozdíl paliva dle tankování (tankoval jsem do vypnutí pistole).
- Průměrná spotřeba a ujetá vzdálenost z palubního počítače.
- Průměrná spotřeba s výpočtem pomoci váhy vzduchu s přihlédnutím na bohatosti směsi z lambda sondy.
- Průměrná spotřeba s výpočtem pomoci váhy vzduchu bez přihlédnutí na bohatosti směsi z lambda sondy.
- Průměrná spotřeba s výpočtem pomocí alternativní váhy vzduchu s účinností motoru 85 % a objemem motoru 2 litry s přihlédnutím na bohatost směsi.
- Průměrná spotřeba s výpočtem pomocí alternativní váhy vzduchu s účinností motoru 85 % a objemem motoru 2 litry bez přihlédnutí na bohatost směsi.
- Výpočet z aktuálního zatížení motoru, s maximálním průtokem 20 l/h.
- Ujetá vzdálenost měřená z řídicí jednotky.

Kromě průměrných hodnot byla zapisována i aktuální spotřeba z palubního počítače a mobilního zařízení při ustálené rychlosti. Byly porovnány hodnoty z úseku cca 150 km. Bohužel nemohly být porovnány údaje z ujetí plné nádrže až po rozsvícení rezervy, protože testovací vůz je „vybaven“ filtrem pevných částic, který během aktivního vypalování zvyšuje spotřebu, a to by mělo za následek zkreslení naměřených hodnot.

6.1 Výsledky měření

Porovnání hodnot průměrně spotřeby můžete vidět v tabulce 6. Nejmenší odchylku oproti palubnímu počítači má měření pomocí zatížení motoru, jenž je vyšší pouze o jedno procento, nicméně palubní počítač ukazuje spotřebu cca o 10 procent vyšší, než jaká ve skutečnosti je. Ke skutečné spotřebě má nejblíže spotřeba vypočtená pomocí váhy vzduchu s přihlédnutím k bohatosti směsi, která je nižší o 6,2 procenta. Průměrná spotřeba dle palubního počítače je pak o 18 procent vyšší než průměrná spotřeba dle výpočtu pomoci váhy vzduchu s přihlédnutím k bohatosti směsi.

V tabulce 7 pak můžeme vidět zápis aktuální spotřeby dle palubního počítače a výpočtu pomocí váhy vzduchu. Informace o aktuální spotřebě nejsou bohužel příliš věrohodné, protože se

hodnoty v čase neustále měnily. Avšak průměrná odchylka z této tabulky je 17,8 procenta, takže jak průměrná tak i aktuální spotřeba vykazuje u výpočtu pomoci váhy vzduchu hodnoty o cca 18 procent menší než hodnoty z palubního počítače. Z testu vyplývá, že pro měření spotřeby je nejvhodnější použít výpočet pomoci váhy vzduchu s přihlédnutím k bohatosti směsi s korekčním činitelem 1,061.

Naměřené hodnoty ujeté vzdálenosti pak můžete vidět v tabulce 8. Ujetá vzdálenost v palubním počítači je sice zaokrouhlována na jedno desetinné místo, nicméně i tak je odchylka nulová.

Tabulka 6: Porovnání průměrné spotřeby.

Zdroj měření průměrné spotřeby	**Průměrná spotřeba (l/100km)**	**Odchylka vůči skutečnosti (%)**
Dle tankování	6,5	0
Dle palubního počítače	7,2	+ 10,8
Dle váhy vzduchu bez přihlédnutí na blahost směsi	5,4	- 16,9
Dle váhy vzduchu s přihlédnutím na bohatost směsi	6,1	- 6,2
Pomoci alternativního výpočtu váhy vzduchu bez přihlédnutí k bohatosti směsi	5,2	- 20
Pomoci alternativního výpočtu váhy vzduchu s přihlédnutím k bohatosti směsi	5,9	- 9,3
Dle aktuálního zatížení motoru	7,3	+ 12,3

Tabulka 7: Porovnání aktuální spotřeby palubního počítače a výpočtu dle váhy vzduchu s přihlédnutím na bohatost směsi.

Rychlost (km/h)	Spotřeba dle palubního počítače (l/100km)	Spotřeba dle váhy vzduchu s přihlédnutím na bohatosti směsi (l/100km)	Odchylka (%)
60	4,4	3,8	- 12,7
80	6,2	4,9	- 21
100	6,1	4,8	- 21,3
130	6,8	5,7	- 16,2

Tabulka 8: Porovnání ujeté vzdálenosti dle palubního počítače a měření z aplikace.

Ujetá vzdálenost dle palubního počítače	Ujetá vzdálenost dle měření z aplikace
165,3 km (zaokrouhleno)	165,321 km

7 Závěr

Původní záměr celé práce bylo vytvoření dvou aplikací, z toho jedna aplikace měla být obsažena v navrženém zařízení, které mělo být umístěno v OBD zásuvce. Z volně dostupných informací o časování komunikace ale nebylo možno vytvořit programové vybavení, proto jsem se rozhodl použít už existující rozhraní, které řeší časování. Logika zpracování, ukládání dat se tím pádem přesunula do služby mobilního zařízení, která komunikuje s uživatelským rozhraním a řídicí jednotkou přes rozhraní ELM327. Z pohledu implementace je největší výhoda použití dostupné databáze a tím je docíleno vyšší spolehlivosti, a dále použití stejné technologie a s tím je spojena teoreticky rychlejší odezva mezi službou a rozhraním.

Spolehlivost aplikace je díky použití osvědčených technologii vysoká. Největším zdrojem problémů může být automatické vypínání služby běžící na pozadí operačním systémem v případě nedostatku systémových prostředků.

Z pohledu užití je největší nevýhoda v tom, že mobilní zařízení musí být neustále ve vozidle, neboť by se pak informace neměřily. Tato nevýhoda je ale kompenzována tím, že stačí mít aplikaci nainstalovanou, mít zapnutý bluetooth a po správném nastavení aplikace se vše měří na pozadí, aniž by uživatel musel cokoliv provést.

Naopak největší výhoda v užití je dostupnost rozhraní ELM327, kde se jeho cena pohybuje od 100 Kč výše, tedy budoucí uživatel nemusí kupovat drahé zařízení.

Hlavní přínos práce je v plném zastoupení palubního počítače s grafickým zpracováním, protože žádna aplikace pro systém Android neposkytovala všechny běžně dostupné funkce palubních počítačů. Aplikace bude volně dostupná v Google Play pod názvem „OBD Onboard Computer“.

Literatura

[1] On-board diagnostics - Wikipedia, the free encyclopedia. *Wikipedia, the free encyclopedia.* [Online] 10. 4 2016. [Citace: 12. 4 2016.] https://en.wikipedia.org/wiki/On-board_diagnostics.

[2] OBD Modes and configuration (PID) - Outils OBD Facile. *Outils OBD Facile.* [Online] [Citace: 8. 2 2016.] http://www.outilsobdfacile.com/obd-mode-pid.php.

[3] OBD Readiness Monitors Explained | OBD Auto Doctor Scantool Garage. *OBD Auto Doctor Scantool Garage.* [Online] 1 2014. [Citace: 16. 3 2016.] http://www.obdautodoctor.com/scantool-garage/obd-readiness-monitors-explained/.

[4] OBD-II PIDs - Wikipedia, the free encyclopedia. *Wikipedia, the free encyclopedia.* [Online] 9. 4 2016. [Citace: 12. 4 2016.] https://en.wikipedia.org/wiki/OBD-II_PIDs.

[5] VAS-5052. *DataScan LB.Net Automative Diagnostic.* [Online] [Citace: 16. 2 2016.] http://www.datascanlb.com/index/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=73&category_id=34&keyword=vas+5052&option=com_virtuemart&Itemid=139 (OBR).

[6] 16# ORGINAL VAS 5051B VW, Audi, Skoda, Seat u.ä. Diagnosegerät. *ebay.* [Online] 26. 3 2016. [Citace: 1. 4 2016.] http://www.ebay.de/itm/16-ORGINAL-VAS-5051B-VW-Audi-Skoda-Seat-u-ae-Diagnosegeraet-/161989213997.

[7] Ross-Tech: VCDS Tour. *Ross-Tech Diagnostic Software for VW-Audi Group Cars.* [Online] 6. 5 2000. [Citace: 26. 2 2016.] http://www.ross-tech.com/vcds/tour/main_screen.php.

[8] Ross-Tech: VCDS Tour: Fault Codes. *Ross-Tech Diagnostic Software for VW-Audi Group Cars.* [Online] 6. 5 2000. [Citace: 26. 2 2016.] http://www.ross-tech.com/vcds/tour/dtc_screen.php.

[9] OBD2 II ELM327 V2.1 Auto MINI Bluetooth Diagnostic. *ebay.* [Online] 1. 4 2016. [Citace: 4. 4 2016.] http://www.ebay.com/itm/OBD2-II-ELM327-V2-1-Auto-MINI-Bluetooth-Diagnostic-Scanner-Tool-for-Car-Blue-/201486908980?hash=item2ee98e3e34:g:iDUAAOSw~otWc3A0&item=201486908980&vxp=mtr07.

[10] **Hawkins, Ian J.** Torque Pro (OBD 2 & Car). *Google Play.* [Online] [Citace: 10. 3 2016.] https://play.google.com/store/apps/details?id=org.prowl.torque&hl=cs.

[11] ELM327 AT Commands. *ELM Electronics.* [Online] 1. 1 2014. [Citace: 16. 2 2016.] http://elmelectronics.com/ELM327/AT_Commands.pdf.

[12] **Lightner, Bruce D.** MAP- and MAF-Based Air/Fuel Flow Calculator. [Online] 30. 10 2013. [Citace: 15. 3 2016.] http://www.lightner.net/obd2guru/IMAP_AFcalc.html.

[13] Smartphone OS global market share 2009-2015 | Statistic. *Statista.* [Online] [Citace: 16. 4 2016.] http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.

[14] Jak vypadá Android uvnitř aneb co je ROM, kernel, bootloader a další? *Android Market.* [Online] 1. 1 2012. [Citace: 16. 4 2016.] http://androidmarket.cz/android/jak-vypada-android-uvnitr-aneb-co-je-rom-kernel-bootloader-a-dalsi/.

[15] Activity | Android Developers. *Android Developers.* [Online] 8. 4 2016. [Citace: 10. 4 2016.] http://developer.android.com/intl/en-us/reference/android/app/Activity.html.

[16] Adding the App Bar | Android Developers. *Android Developers.* [Online] 2016. [Citace: 3. 2 2016.] http://developer.android.com/intl/en-us/training/appbar/index.html.

[17] STK zpřísnily: u emisí budou potíže, pozor na úpravy - iDNES.cz. *Idnes.* [Online] 21. 1 2016. [Citace: 14. 3 2016.] http://auto.idnes.cz/prisnejsi-kontroly-stk-v-praxi-d33-/automoto.aspx?c=A160119_223823_automoto_hig.

[18] Number of apps available in leading app stores as of July 2015 | Statistics. *Statista.* [Online] [Citace: 16. 4 2016.] http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.